APPENDIX A

Introduction

Appendix A will be understood and appreciated from the following detailed description, taken in conjunction with the drawings in which:

Fig. 43 is a simplified block diagram showing relationships between data sources and a processing module constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 44 is a simplified block diagram showing relationships between data sources and a pre-emptive processing module constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 45 is a simplified block diagram showing user defined tasks of a processing module, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 46 is a simplified block diagram illustrating a composite processing module constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 47 is a simplified block diagram illustrating input channels into a pre-emptive processing module and a cooperative processing module, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 48 is a simplified block diagram illustrating output channels from a pre-emptive processing module and a cooperative processing module, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 49 is a simplified block diagram illustrating a registration processing module constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 50 is a simplified block diagram illustrating a root process manager, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 51 is a simplified block diagram illustrating an execution graph scenario constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 52 is a simplified block diagram illustrating a root directory tree from which the SIP begins, constructed and operative in accordance with a preferred embodiment of the present invention; and

Fig. 53 is a simplified flow chart illustrating SIP states and the commands that effect them, constructed and operative in accordance with a preferred embodiment of the present invention;

Presented is an architecture of the kernel of the SIP (software image processing) that is capable of running efficiently on SMP (symmetric multi-processing) machines (e.g., a multiprocessor Intel based machines), and on a single processor machine. The implementations of object oriented techniques to implement multiprocessing in a platform independent way is discussed.

The goal of the SIP kernel is to allow easy and efficient processing of different scan scenarios. A scan scenario usually involves connection of the scanner's hardware, reading incoming information about the scanned board and processing this information in different ways. For example, in a learn scan scenario we create a reference of the board from the incoming data, and while in the inspect scenario we compare this reference to the actual data coming from the scanner. If the inspected board is too different from the reference board, we decide that the board is defective. The scan scenario may involve only off-line computations, without connecting to any scanner. We may look at the SIP as a tiny operating system that is responsible for carrying out efficiently many different scan scenarios.

The SIP schedule is a set of processing modules. Each processing module consumes zero or more data sources and produces zero or more data sources. A data source is any data structure shared by more than one processing module.

In this appendix changes to two basic entities in the SIP - the task/channel and the data source - are discussed. The separate task and channel objects are preferably joined as a new "meta-task" object which can function both as a task and as a channel. The channel capabilities of this new object are provided by a new kind of data source which encapsulates the access to data via hardware - the Stream_data_source. A preferred modification to the data sources is the capability to allocate a data source in shared

memory, thus allowing "meta-tasks" executing in more than one process to share data efficiently and transparently.

Definition of an abstract processing module

This abstract base class represents the most basic computational element that can be defined within the SIP. A Processing module preferably comprises a manager which is a method/program that initiates and manages it.

Reference is now made to Figure 43 which is useful in understanding a preferred embodiment of the present invention.

A processing_module may have up to three kinds of "relationships" with data sources:

Consumer of a data source (reader) - reads and processes data from these data source(s).

Asynch Consumer of a data source (reader) - the processing module registers to be notified by the data source whenever new data is available for reading and processing.

Producer of a data source (writer) - writes produced data to produced data sources.

A processing module typically has at least one consumed data source to be correctly configured. If a processing module does not consume any data source, it will not get scheduled to process.

The basic Processing Module performs some sort of processing on consumed data. Following are several examples of concrete derived classes (which are discussed in greater detail in the following sections):

Some sort of computation performed on input DS (data source) with results written to the output DS.

A parsing operation on input data, with parsed data written to selected data source.

A formatting operation on input data, to write it to an output stream data source.

The Processing Module is defined in the SIP configuration file.

Another basic trait of the abstract base class is now defined:

Definition of a pre-emptive processing module versus cooperative.

Reference is now made to Figure 44 which is useful in understanding a preferred embodiment of the present invention.

A pre-emptive processing module (denoted by the broken lines in Figure 44) exists within an independent operating system process/ thread. As such, it is scheduled to be run by the operating system according to system level events. A cooperative Processing_Module is scheduled by it's manager, usually a higher level composite processing module scheduler.

All data sources referenced by a pre-emptive processing module are preferably placed in shared memory, denoted by the broken lines. This is preferably done automatically by the configuration file loader.

Indication of data available for processing/ end of data etc. is typically signalled in the following manner by the updated data source:

For pre-emptive processing modules, communication will be through pipes which are to established by every pre-emptive processing module.

For a cooperative processing module active signalling is not typically needed. The update of more data ready to process.

Every pre-emptive processing module processing module has a unique ID which is its operating system process ID. To this ID the file descriptor associated with the pre-emptive process pipe is attached. Data sources which are dependent on streams external to the SIP will also provide an external file descriptor to which the owning pre-emptive processing module will be attached.

The attachment of file descriptors to pre-emptive processing modules is performed after the load configuration process. This may be a portability issue because there are platforms which allow only one un-named pipe per processes.

The consumer registration process preferably works as follows:

The consuming processing module calls Register_Consumer() of the data source, providing the data source with its pipe object.

The data source method Register_Consumer() will return a pipe object on which the consuming processing module is to sleep. If the data source is an internal SIP data source, it will return the same pipe object with which it was called, indicating that it will send a wake up call to the registered consumer's wake up pipe. If the data source is a representation of an external data stream, it will return a different "pipe object" to which the external file descriptor is attached.

The Register_Consumer() method accepts a grain size parameter, which determines how often data available notifications will be sent. If a gain size of zero is selected, than notifications will get sent only for the following events: The data source has reached end_of_data status (end of scan), or by an explicit notification call made by a producer.

Data sources produced by a processing module now fall into two categories - shared and local. Writing data to a local data source is effected without making changes. It gets more interesting when writing data into a shared data source: if a pointer to data allocated by the task on a heap is provided, then it needs to be ensured that data is allocated on the shared memory heap, otherwise the pointer will be invalid in the context of a different pre-emptive processing module.

When a data source determines that it has sufficient data for filling a data grain as registered by one of it's consumers, it sends an alert to the consumer. The alert does nothing if the consumer is a local task, and sends a notice to the consumer pre-emptive processing module's pipe.

A data source's producer may call a flush() method of the data source which will cause it to wake up all of its consumers. This will occur either at end of scan or when the processing of a big chunk of data has been completed.

Concrete Derived Classes - a sample taxonomy

Reference is now made to Figure 45 which is useful in understanding a preferred embodiment of the present invention.

Following are several examples of processing modules:

The user defined task may be of several types:

Composite processing Module - Execution graph - Some sort of computation performed on input data with results written to the output, and two types are typically available - cooperative and pre-emptive.

Input Channel - A parsing operation on input data, with parsed data written to selected data source. The input data will come from an input stream data source, which will be attached to either a file, TCP/IP or a DMA (Direct Memory Access) channel. Writing to a selected data source will allow step and repeat processing in the future, with data from different repeats dispatched to different output data sources which each serve as input to independent execution graphs.

Output Channel - A formatting operation on input data, in preparation to writing the input data to an output stream data source. The output stream may end up in a file or in a remote computer via a TCP/IP connection.

Registration Transform Producing Task - A computational task which consumes input scan data source non-Asynchronously and writes both an updated transform to a scan line transform data source and excess-missing reports to an excess-missing data source.

A detailed description of these processing module derived classes follows.

Definition of a composite processing module

Reference is now made to Figure 46 which is useful in understanding a preferred embodiment of the present invention.

A composite processing module typically consists of several processing modules which are mostly interconnected between each other. All data sources which are used locally only are placed in local scope and their names are not available globally.

Only data sources which are external to the local processing modules are connected to the external scope processing modules. The connection is maintained through the composite processing module which schedules the internal processing modules when notified by the external data sources.

Because a pre-emptive processing module cannot be scheduled by a SIP scheduler object, a composite processing module may not have pre-emptive processing modules in it's execution graph.

Every processing module may be a composite (and thus have many processing modules "embedded" in it). A preferred limitation is that all "embedded" processing modules of a pre-emptive composite processing module are typically cooperative so that they may be managed by it.

This limitation defines a constrained general processing module structure: all of the first level processing modules are preferably pre-emptive, and all these embedded in a first level composite pre-emptive processing module are preferably cooperative.

An input channel

Reference is now made to Figure 47 which is useful in understanding a preferred embodiment of the present invention.

An input channel from DMA is constructed by coupling a local Input Stream data source as the single consumed data source of a processing module parsing task.

When the processing module is pre-emptive, the parsing task is awakened to work only when input data is available in the stream data source, which is local to the context of the input channel process. When the processing module is cooperative it will work continuously on the input stream.

The input channel process sleeps both on the command pipe (for commands from the root SIP process) and on the underlying stream system level File Descriptive.

This design assumes that whenever data is available on the input stream, the channel process will read as much data as possible, parse it and write it to the produced data sources. This design works well when the input stream is based on an external data provider such as either DMA or a TCP/IP stream.

When working with a local file, reading the entire file in may cause the produced data sources to overflow. Obviously, working with a local file typically requires some scheduling beyond that provided by the operating system.

The solution is to place local file input stream data sources within a cooperative composite processing module which will schedule reading of data in well sized chunks. This solution will work well, since a local file may be considered synchronous and typically does not require a separate operating system process to handle access to it.

An output channel

Reference is now made to Figure 48 which is useful in understanding a preferred embodiment of the present invention.

An output channel is constructed by coupling a local Output Stream as the single produced data source of a processing module which formats (or serializes) it's input Asynch shared data sources into the output stream.

The processing module may be either pre-emptive or composite:

A pre-emptive processing module typically is used when the output stream is directed to some "blocking" system device such as a TCP/IP connection, or when output to a file is under very low priority and is to be scheduled only when the system is idle.

When the stream is directed to a regular file, the operating system is preferably able to handle the write requests without blocking the writing process. Therefore, an Ostream data source which writes to a file could be placed within a cooperative composite processing module without significant loss of performance (except that related to the actual write to file).

A Registration Transform Cooperative Processing Module.

Reference is now made to Figure 49, which is useful in understanding a preferred embodiment of the present invention.

The registration task is implemented as a "Registration processing module", which may be implemented as a pre-emptive or cooperative module. This task consumes a scan line data source, and produces a transform scan line data source and an Excess-Missing data source.

When the registration task is cooperative, it is implemented within a composite processing module and is scheduled to process data by it's composite owner's scheduler.

When the registration task is pre-emptive, it sets up a either timer which awakens it at timeout or defines a large data grain for notification and wake up.

Definition of the Root Processing Module (the Process Manager).

Reference is now made to Figure 50, which is useful in understanding a preferred embodiment of the present invention.

The root process has three objects which are activated by a main loop which is entered after the SIP has been configured by the application. Prior to configuration, the only active object is the interface to the application. The SIP typically has at least one of the two processing objects - either a Processing Module Manager and/or a local composite cooperative processing module.

The three objects are:

Interface to the application - an object which handles all communication and parsing of the commands sent to the SIP by the application.

Local single composite cooperative processing module. This module is made available in the root process for the following reasons:

* To allow easy debugging of new execution graph modules by developers. This is by letting a complete execution graph run in one debuggable process.

* For allowing the running of "stand-alone" SIP programs (such as a background learn- update).

* Backwards compatibility during the development phase of multi-processing capabilities.

Processing Module Manager - an object which handles all communication with the stand alone pre-emptive processing modules.

Every pre-emptive processing module sets up one pipe on which it listens for messages from the other processes in the SIP group of processes. The root manager process sets up a similar pipe to listen to acknowledgements/ messages from the managed pre-emptive processing modules.

The following kinds of messages may pass in the system between the processes in the group:

Stop/Status Query commands from the root processing_module manager to the pre-emptive processing modules.

Command acknowledgement and error state/ end of scan reporting back to the root process.

Indication of data gain availability/ end of data/ error in data in a shared data source - the data source notifies all consumers which had registered with it for notification.

The message is typically passed as a stream of characters. The basic message format is:

Field Position [char]	Field Type	Description
1	message character	Message represented by character: commands from root: 's' - stop scan; 'q' - query status (ping); 't' terminate child process; 'r' - reset to initscan condition, 'u' - do uninitscan, 'b' - start scan, 'f' - go to endscan(ok), 'g' - endscan(error) acknowledgement to root: 'k' - child killed; 'x' - child in error mode; 'v' - child running; 'i' - child idle in scan } child in INIT state 'e' - child at end of scan state 'y' - child in OK state. message from other SIP process: 'd' - notification of change of status in a registered consumed data source to pre-emptive consumer.

2:6	sender SPID	The message sender's system process ID in ascii (00000-99999)

The Root process will typically "barrier sync" all child processes so that when a new command is issued they all start out in the same state.

Data Source Modifications

There are two main changes which are to performed on current SIP data source objects.

One change is typically used by selecting Multi-Processing - the capability to allocate a data source is shared memory which is visible in all the SIP group of processes. The data source preferably implements the minimal level of access protection/ mutual exclusion used by the current spec for access to this data source.

The second change is typically used due to the merging of the single process SIP task and channel objects to the "Processing_module" object - the adding of stream data sources.

Also, the helper classes Sip_mem, chain_of_blocks and reservoir which manage memory (Such as a memory model for a DS_Array type of data source) is preferably shared memory compatible. This compatibility means:

1. ALL allocations are typically made in the shared memory heap.
2. Mutex / Reader-Writer access locks - to be used sparingly as possible, with a single writer- multiple reader atomic write capability allowing no mutex's to be utilized.

Data sources in shared memory

The allocation of data sources in shared memory is typically directed by the SIP class configuration file reader, which selects the correct allocation mode according to the context.

Data sources placed in the top-level configuration file are typically allocated in shared memory. When loading configuration files for subsequent composite Processing_module (s) the data sources are preferably built locally..

A data source is typically created in shared memory by the sip_factory by specifying an optional second parameter to its *clone* method as "true". The parameter defaults to false for default objects constructed on the local heap.

The writer of a data source preferably provides an extra implementation of a shared memory creator method;

A task which interacts with a shared data source (and provides the data source with pointers to data which the task allocates) typically checks if any of the data sources are in shared memory, and allocates its data accordingly.

A shared memory data source preferably allocates all its dynamic storage on the shared memory. This does not necessarily mean that a different implementation is used for a local heap/ shared heap data source. Some alternative options include:

1. Turn current data source into an abstract class, and provide two derived classes - one for shared memory and one for local. Define two not valid methods in the derived classes to return errors. This forces the execution graph creator to specify a different kind of class.

2. Allocate data source to dynamic allocations in shared memory, which then allows the user to safely use the interface described above to select shared / local data sources. This using the shared heap may be slower than using the local heap, due to the mutex lock used by the shared new operator to protect its own internal data.

Following from the previous solution, foregoing direct memory allocations and using private memory pools which manage the shared/ non shared issues is probably the most robust solution.

A third approach, which seems more robust and user friendly, is selected for the initial implementation.

It is appreciated that shared data sources may get pointers to data allocated by the producing task. Any data which is passed this way is preferably allocated in shared memory by the task. Tasks are preferably checked to see if they are shared memory compatible. An example of an item which is affected by this is the memory_model which allocates and handles data for ds_array class objects.

Stream data sources

A stream data source is an encapsulation of a SIP i/o utility object, which provides it with an interface compatible with a generic data source. This interface allows the coupling of a Stream_data_source object to a processing module and the creation of an I/O (input/output) channel.

The basic abstract class would be Stream_data_source which is inherited from data_source. This basic class provides a generic I/O wrapper for io_util. Concrete derived classes include:

IStream_data_source - An encapsulation for an input from FILE, TCP/IP or (future) DMA.

OStream_data_source - An encapsulation for output to a file or TCP/IP.

Step and Repeat Considerations

A preferred method for handling step and repeat situations is to define the small reference data base, create several execution graphs (the number being defined by how many repeats can be handled in parallel) and to have the input channel send the incoming scan data to the execution graph corresponding to a certain repeat.

Reference is now made to Figure 51, which is useful in understanding a preferred embodiment of the present invention.

The execution graph scenario typically appears as in Figure 51 with at least five pre- emptive processing modules. If a decision is made to split up the per-repeat execution graph, then there typically will be eight or more pre-emptive processing modules.

The logic typically used to handle the step and repeats are typically incorporated into *three* modules in the system:

The root process is preferably aware of the current scan data repeat number, and which execution graphs are busy on which repeats. The root process manages the execution graph priorities so that the oldest scan still processed will be of the highest system priority.

The input channel pre-emptive processing module will know how to detect a change in repeat number from the incoming scan data, and switch the output of parsed scan data into the appropriate data sources. It typically selects the data sources according to the table of free execution graphs maintained by the root process.

The output channel pre-emptive processing module preferably knows which execution graph is processing which repeat, in order to buffer and format its output of data to the application.

End of scan is typically handled in a slightly more complex way: An end of scan condition reported by an execution graph will typically be interpreted as "end of repeat processing" by the root process, which then sends the execution graph a "InitRepeat" message which will reset it for handling the next repeat. The SIP typically switches to end of strip-scan condition when the input channel pre-emptive processing module signals "end of scan" and after all the execution graphs and the output channel pre-emptive processing module have signalled "end of scan".

In a more complex scenario, several kinds of execution graphs may be supported concurrently: e.g., in a learn - scan -scan ..... scenario. The above mentioned architecture scales well to this preferred requirement: The table of free execution graphs is preferably extended to hold the type of execution graph, so that any number of classes of execution graphs can be handled by the root process.

Detailed Class Definitions:

Documentation is typically generated on-line automatically for various classes (Fig. 53).

APPENDIX B

Appendix B will be understood and appreciated from the following detailed description, taken in conjunction with the drawings in which:

Fig. 54 is a simplified illustration of a single snapshot image (referred to herein is a "Snap") covering the area around a single Color_defect report, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 55 is a simplified illustration of snaps covering the area around multiple Color_defect reports, constructed and operative in accordance with a preferred embodiment of the present invention; and

Fig. 56 is a simplified schematic illustration of the structure of a line of 48 snap reports, constructed and operative in accordance with a preferred embodiment of the present invention.

A Snap is a rectangular or modular-rectangular region that contains RGB (red, green, blue)-information of the pixels of that region. The RGB-information is collected from a certain area that surrounds a report of the type Color_defect. The minimal region in which the RGB data is collected around a single report is of size n x n (pixels). In the case of the present embodiment n = 48.

Reference is now made to Figure 54 which is an illustration of a single snap covering the area around a single Color_defect report, which is useful in the understanding of a preferred embodiment of the present invention.

Single snap report

The single snap report is a 32-bit word. Its internal structure is the following:

8 bit : x_s - x coordinate in 8-bit representation.

8 bit : r - value of red.

8 bit : g - value of green.

8 bit : b - value of blue.

The 8-bit representation of the x-coordinate is faithful provided that the snap reports in a given line are ordered by their 12-bit value of the x-coordinate. The conversion from the 8-bit representation of the x-coordinate to a 12-bit representation is explained below.

The origin and structure of snap data

Snaps are typically created "on-line", during a scan, for example during the inspection scenario. Each slice is treated separately and is divided into basic block units. The basic unit is a square of size 16 x16. Each Color_defect report leads to the recording of 9 blocks of RGB data: the block which contains the Color_defect report itself and its 8 neighboring blocks, as illustrated in Fig. 54.

A snap area that is recorded due to several Color_defect reports, is determined according to the following rules:

1. The minimal snap is typically of size 48 x 48 $(pixels)^2$, and it consists of nine blocks of size 16 x 16 $(pixels)^2$ each.

2. Each snap report preferably appears only once, even if it was originated due to several Color_defect reports (see Figure 55).

Snap reports are typically ordered in lines, according to their y and x coordinates. Each line contains a sequence of 16*N single snap reports, accompanied by information about the slice in which the reports were originated, and the number of the reports in the line. The reports in the are divided into N groups of 16 reports all of which have the same value of x in the 8-bit representation. Fig. 56 shows schematically the structure of the line of snap reports, with the minimal value of N (N=3).

The 8-bit to 12-bit conversion of the x coordinate

The conversion from the 8-bit representation to the full 12-bit representation of the x coordinate is done in the following manner:

• Let x_s be the x-coordinate in the 8-bit representation of a given snap report.

• Let n_b be the number of the snap report within the relevant group of 16 reports, all carrying the same 8- bit value of x_s. The numbering is done in the following way: the first report in the group has $n_b = 0$, the second $n_b=1$, etc.

Then the 12-bit representation of the snap report's x-coordinate is given by:

$x = x_s * 16 + n_b$

Ds_array<Snap>

Ds_array<Snap> contains Snap reports ordered by their y and x coordinates. The structure of the single Snap report is as mentioned earlier. The x coordinate is given in

8-bit representation, and is preferably converted to a 12- bit representation. As in the case of all data sources, a pointer to the beginning of each ds_array_line is available, as well as the information about the slice in which the data source was originated .

APPENDIX C

1. PIM-FILTER

Appendix C will be understood and appreciated from the following detailed description, taken in conjunction with the drawings in which:

Fig. 57 is an example of a BGA (ball grid array) board to be inspected by a system constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 58 is a close-up view of the lower end of the BGA board of Fig. 57;

Fig. 59 is a close-up view of a bond finger on a BGA board, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 60A is a flow chart illustrating data flow through a PIM map generator during the Learn stage of an inspection scan, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 60B is a flow chart illustrating data flow through a filter during the Inspect stage of an inspection scan, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 61 is a simplified block diagram illustrating subtraction of regions by the map generator, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 62 is a simplified block diagram illustrating the data structure of a PIM stored in runlength format, constructed and operative in accordance with a preferred embodiment of the present invention; and

Fig. 63 is a simplified block diagram of the general structure of the PIM system, constructed and operative in accordance with a preferred embodiment of the present invention.

1. Introduction

The scanned area of an inspected object, such as a BGA panel, contains regions of varying importance, ranging from sensitive bonding regions and up to text regions that are to be ignored. The Image Processing procedure should treat each region with different criteria of sensitivity in order to obtain maximal detection and minimal false alarms. Two concepts of differentiating between areas are possible: filtering out information in non-interesting regions, and region-dependent type and sensitivity of inspection.

In accordance with a preferred embodiment of the present invention, image processing is composed of several phases: defect detection, defect verification, and top-down windows. The defect detectors are realized both in hardware, like the nick-protrusion and surface defects channels, and in software, like the excess-missing and line-width channels. Each of these detectors produces suspicious events that might, or might notsignal, real defects; some of events may be certain defects and preferably need to be verified by a more careful inspection such as by CEL-to-CEL analysis as described with reference to Figs. 30 - 30C in the specification. Another method of detecting suspicious events is by preparing in advance (during the Learn scan) top-down windows in areas of special interest, and examining them more carefully during Inspect.

In this appendix, two, apparently different, methods that realize region differentiating are discussed:

- Filter - screens events according to their type and position, ignoring irrelevant areas such as text areas, areas known as "unstable", etc. Placing the filter at a critical position in the IP pipeline would also save slow processing of non-interesting areas and increase speed and efficiency.
- Tolerance Setting - specifies the accuracy of inspection test of events, according to their type and geometric location with respect to regions of interest.

Both these methods rely on the same geometric query tool, namely Point In Map (PIM), that answers the following question: which region in a map contains a point (event).

FIGURE 57. is an example of a BGA board. The thick black regions are exposed metal and the stippled regions and thin line regions are metal covered with solder mask. The exposed metal regions are important and preferably are inspected carefully, while covered areas, typically, are less critical.

2.1 Filter

2.1.1 Scope

The Filter, which will be realized as a SIP task, receives as input a list of events and produces a new list that contains only the "interesting" events. The filtering criteria

are geometry and type, and can be either set by the user or automatically constructed during the Learn process.

Filtering is performed in several levels: filtering of raw data, namely CELs and features, filtering of events suspected as defects resulting from the defect detectors, and filtering of certain defects resulting from the verifiers.

Filtering raw data has two main drawbacks: it can be time consuming since it is realized in software, and in the case of CELs it generates "holes" in the incoming data, which can be harmful to other processes such as the connected components application. Filtering of suspected events is less damaging to the picture and reduces substantially the amount of features for filtering, however it compels redundant processing of ignoring areas by the defect detectors. Filtering of verified defects compels verification, which is a long and difficult process, of excessive events and thus is inferior to the other two.

The Filter uses the PIM geometric tool for map generation and point location queries. The format of input points is scan line format, ordered lines of events in increasing y-coordinate. This can be used to increase the efficiency of search by using the knowledge of the former point. More details are discussed in the PIM description in section 3.3.

FIGURE 58. is a close up of the lower end of the BGA picture, with a region to be ignored being marked.. The marked region is misalignment between the solder mask and the metal which is not a defect, yet might cause false calls.

2.1.2 Input

The input of the filter task includes the following features:

- filtering map - a planar subdivision of the scan area is generated by the PIM according to the filtering regions, where every point (x,y) is contained inside a single face. Each face represents a different filtering criterion, namely type(s). The map is given in "ideal" reference coordinates, meaning after adjustments of pixel size, shape, etc. If the scan area is composed of several duplicates of the same pattern, namely step and repeat, only a single-repeat map is needed.
- registration - an alignment transformation between the ideal Learn coordinate system (filtering map) and the ideal Inspect system (on-line features). If no registration

is given, as in the case of design-rule defects computed during the learn inspection process, the unit transformation is taken. The allowed transformation is affine, namely shift, rotation, and expansion.

- events - a list of events in scan line format, characterized by type and position.

2.1.3 Output

- events - a partial group of the Input list of events in on-line coordinates, that contains only the ones that are not filtered.

2.2 Tolerance Setting

2.2.1 Scope

The verification test function performs an accurate measurement of "suspected" defects and decides, according to previously defined sensitivity criteria, whether it is a real defect or not. The sensitivity criteria depend on the exact location of the defect. Thus a geometric location tool is needed to determine the location of the defect and hence the accuracy of measurement necessary for verification.

The Tolerance Setting is a service function (func_defects_handler) that is used either before or after the actual test function is applied. If the type of test depends on the accuracy of measurement, this service is called before the test in order to determine its type. If the type of test does not depend on the allowed tolerance, this service is used for sensitivity setting. It should be noted that one of the test functions possible is filtering. In this context filtering is performed for arbitrary positioned events, hence falling outside the Filter scope.

The Tolerance Setting function asks the same geometric location queries as the Filter, however the format of input events is position random instead of ordered. The PIM tool gives services of map generation and point queries also to the Tolerance Setting application.

FIGURE 59 is a close up of a bond finger as an example of varying inspection sensitivity. Region 1 is unstable, meaning its exact location is may change, because of the edge of the solder mask. Region 2 is a critical bonding region. Region 3 is non-critical. Hence region 2 is the most sensitive, and then in descending order regions

3 and 1.

2.2.2 Input

• sensitivity map - a planar subdivision of the scan area generated by the PIM according to interest regions, where every point (x,y) is contained inside a single face. Each face represents a different defect sensitivity, namely allowed tolerance, represented by some type. The map can cover only part of the scan area, such as a trigger window or a single repeat in step and repeat panels. The map is given in ideal reference coordinates.

• registration - an alignment transformation between the ideal Learn coordinate system (sensitivity map) and the ideal Inspect system (on-line features). If no registration is given, as in the case of design-rule defects computed during the learn inspection process, the unit transformation will be taken. The allowed transformation is affine, namely shift, rotation, and expansion.

• event - "suspected" defect or trigger for inspection, in arbitrary position, with type of test function to be applied.

• mask table - a Look Up Table (LUT) that associates the map types to a different inspection sensitivity.

2.2.3 Output

• inspection accuracy - sensitivity of inspection, such as maximal allowed nick size, that will determines whether a "suspected" defect is real or false.

Design

3.1 Filter

The filter includes the following operations:

1. Take a row or a set of rows of events.

2. Transform the events via registration to the Learn coordinate system. Every several rows the transformation is updated by a new dynamic registration.

3. Send each event and the filtering map to the PIM for a position and type query.

4. Receive from the PIM a positive answer for a position and type match between the event and the map, and a negative answer otherwise.

5. Filter out positive events.

6. Inverse transform the remaining events back to the on-line coordinate system.
7. Generate a data source containing the events that are not filtered.

3.2 Tolerance Setting

Outline of the tolerance setting service:

1. Receives a trigger, or a "suspected" defect + type of defect.
2. Apply registration to convert to Learn coordinates.
3. Send the trigger and the sensitivity map to the PIM for a position query.
4. Receive from the PIM a type denoting the face that contains the trigger.
5. Return to on-line coordinate system.
6. Match an inspection tolerance to the trigger+type according to the mask table.

PIM - Point In Map

3.3.1 Scope

The PIM is a geometric tool designed for two distinct functions: generate a unified map from a set of regions, and locate a point in that map. The Map Generator creates a map, where each face is associated with a set of types. The Location Query determines which face contains the input point and either returns its type(s), or determines whether it matches the point's type and returns a boolean answer.

Editing and Display of the map are not in the scope of the PIM, and are the responsibility of the User Interface Application. Nevertheless, separate tools that support those functions will be developed for simulation purposes.

The two functions of the PIM are demonstrated in the following data flow sketch:

FIGURES 60A - 60B are Examples of data flow during an inspection scan: (60A) map generator during Learn. (60B) Filtering during Inspect.

3.3.2 Map Generator

3.3.2.1 Input

The input for the map generator is a set of regions of general geometric shape, each identified by a single type or a union of types. The regions may be user defined or be learned automatically during the Learn scan according to areas of interest: The PIM accepts regions with the following properties:

- Three possible formats: polygons (a set of points), circles, and run-length.
- Closed and simply connected (for polygons only).
- Overlap is allowed. The type set of the overlap area is determined by the user, and can be the union of the original overlapping type sets, one of the original sets, or any other combination.
- Subtraction of regions is allowed. These "negative" regions are regions with no type, usually defined inside regular regions in order to simplify generation of complex areas. In that case the order of definition is important, namely the latest defined region overrides the former. The negative region is identified by the type to be subtracted from the former defined positive regions. The subtraction operation can also be determined by the user: difference of type sets, symmetric difference, etc.

FIGURE 61. (a) is an example of filtering regions. (b) Importance of definition order: negative region #2 overrides positive region #1, while #3 overrides #2.

3.3.2.2 Output

The output is a unified map of all the regions, resulting in a planar subdivision of the area where every point (x,y) is contained inside a single face. Each face represents a different region or an intersection of regions, and is associated with either a single type or a union of types. The map can cover the whole scan area, or just a part such as a single repeat or a trigger window. The format of the output picture should optimize the storage space, difficulty of realization, and efficiency of location queries for the different input point formats. The first excludes raster and encourages some compressed format. The second supports run-length format, namely a format in which a run of pixels having the same values are represented in a compressed form, rather than a polygon format, since the definition of regions may come in both formats and converting a run-length picture to a set of polygons is significantly harder than the opposite. The third, efficiency of location queries, also supports the run-length option over the polygon one. To conclude, it seems that run-length is the preferred format for the output picture.

3.3.3 Location Query

3.3.3.1 Input

- unified regions map - a planar subdivision of the scan area, each face associated with type(s), in run-length format.
- event - a point associated with a type.

3.3.3.2 Output

There are two types of location queries that produce different output:

1. boolean answer to a location and type match question.
2. set of types that are associated with the face containing the point.

3.3.3.3 Point Location Methods

Point location methods vary according to the exact format of the input points. Clearly, efficient search of an arbitrary point and an ordered set of points are different. First, an arbitrary point search is supplied, and a more efficient method for rows of data is added if necessary.

The fastest and most convenient location of a point can be performed by converting the regions map to a raster format, thereby reducing the query to a random access operation. The main drawback of course resides in the size of such a picture and memory access handling. Next is binary search in a run-length picture which is slower, of the order of log(n) where n is the length of the run-length average row, however it is much more efficient in terms of picture size. If the complexity of pictures is high and the speed of the binary search is not enough, another option is available: addition of an indexing system (Indexing of Ordered Data: SRS) to the run-length picture, enabling random access to small "zoom" areas of the run-length. This system enlarges the stored picture by the additional indexes, however reducing the average row length and hence log(n).

Data structure of PIM

The PIM is stored in runlength format, illustrated in Figure 62.

Additional tools

The PIM is a generic geometrical tool. However, the SIP has specific requirements such as configuration files handling, interpretation of region types, etc.

For that purpose some additional tools are needed as detailed in the next sections. The general structure of the whole PIM system is illustrated in Figure 63.

3.4.1 PIM SIP

PIM SIP contains a PIM and has 2 additional features:

- a translation table from string-types to integer-types.
- a self building method from a regions' file.

The translation table is in order to keep the efficiency of integer-type PIM while providing the user with string-type, meaningful, region identification. The configuration file produced by the user is written in terms of regions' names, and the PIM SIP translates automatically to internal integer-types.

The self reading method provides the other SIP applications an easy interface for constructing a PIM. If the proper file exists (see section), no knowledge of PIM building methods is needed and the PIM SIP will read the file and build itself.

3.4.2 Query Table

A translation table of regions' type to detailed regions' information. A query table is attached to a PIM, and translates its id's (of integer type in case of PIM SIP) to objects that hold quantities related to that region. For example in the case of nick threshold values: maximal allowed width and length. One PIM can be attached to many query tables, each representing a different aspect of interpretation of types: nick/protrusion thresholds, width defects, etc. Query table is templated to the data-objects, that can be one of the existing objects (NickProt, LWdefect, ExcessDif) or user defined.

3.4.3 "MIM"

Manager of a system of PIM and attached query tables. Main function is loading configuration files and construction of PIM plus query tables. Query methods also exist for extracting information of point in map, and more detailed regions' information.

3.4.4 Shape File Handling

A special shape file format, with file reader and file writer to support it, is provided for the purpose of transferring shape information between learn and inspect scans. The file reader/writer uses the double-type shapes of the SIP (Dpolyline, Drect, Dcircle). A Shape_base class, which all shapes inherit from, was also added. Shape_base inherits from Base_factory so that all shapes can be cloned when read from file.

File handling is a separate unit that can be replaced easily by a different format file handling, providing similar methods. Methods include open/clear, read/write, and shape information.

APPENDIX D

I Introduction

A camera model, as preferably implemented under the SIP in the Integrated Circuit packaging (ICP) project, is described. Two main issues are preferably resolved by the model: (a) any geometric information derived from the scanned image is preferably compensated for the distortions evolved in the image acquisition process (this is accomplished by the alignment transformation) to align the data, and (b) since several cameras acting together result in distributed and redundant data, merging and "stitching" of data is typically used.

The classes and tasks supporting the management of several cameras in the system are described, particularly new classes and tasks designated in specific to support the camera model, whereas other classes and tasks which have only a partial role in supporting the management of data in a system of several cameras, are mentioned in less detail.

2 The model

The following is a discussion of the integral components of the system:

2.1 Classes 2.1.1 Coordinate System (CoordSys)

CoordSys structure is a fundamental component of the camera model, as it preserves the integrity of coordinate data in a system of several cameras. A coordinate system representation is attached to many different items in the system; transformations (AffineSdtrana), data sources (Data-source), tasks (Sip-task), SIP data (the many Sipdata structures) and windows (Sipwin). In order to accurately reflect coordinates throughout the system processing, coordinate systems need to go through the proper transition rules whenever they are changed. Integrity is preserved by endowing any transformation method, in specific, the Transform method, which transforms an object from one coordinate system into another, with a transition rule which defines accurately the change of coordinates: Domain coordinate system -> Range coordinate system. In addition, a simple sanity check is done to make sure that the Domain coordinate system equals the coordinate system of the object to be transformed. Moreover, for a compound object, all its data members are in the same coordinate systems (e.g., all data sources of a task, or all sipdatas within a window need to be in the same coordinate system).

The CoordSys structure has three fields:

• type - can have one of the following values { Unknown, Camera, Aligned}.

• context - the context of the coordinate system represented by this object, can have one of the following values { None, Reference, Online }.

« camera-id - If Camera coordinate type, this indicates the Camera id number.

1. 1. Many items in the system are transformed by their Transform method

2.1.2 Single camera environment (Single-camera)

Single-camera class handles information on the parameters defining the environment of a single camera. To mention only the integral ones: pixel size, camera-width, the camera-coverage region (i.e., the quadrilateral aligned region "seen" by that camera), the (Affine) transformation that converts the camera coordinates into aligned coordinates, and the parameters that can be derived from it (e.g., the camera's orientation with respect to the direction of scan). This class provides several methods of inclusion tests indicating whether or not a given geometric object is embedded in the camera's coverage region, and whether or not it resides within an overlap region of the camera. To support a fine discrimination among the several different geometrical relations a geometric entity can have with respect to the camera's coverage region, Single-camera provides the WinGeomInclusion status which can be one of the following:

• ALL_tNSIDE - Entity is inside camera not intersecting any overlap region.

• ALL_IN - Entity is inside camera and intersecting an overlap region.

• CLIPPED_SHARED_LEFT - Entity is clipped against edges covered by a left overlap region.

• CLIPPED-SHAREDJR.IGHT - Entity is clipped against edges covered by a right overlap region.

• CLIPPED_N'OT_SHARED - Entity is clipped against edges not covered by an overlap region.

• ALL-OUT - Entity does not intersect the camera region.

• NOTIN-A-CAMERA - The coordinate system of the entity does not correspond to a camera.

• UNKNOWN

2.1.3 System of several cameras (Multi-cameras) The Multi-cameras class handles information on several cameras.

• Purpose: Handle information on several cameras.

• Methods:

- Put/Get number of cameras to handle.

— Put/Get a single camera information (delivery of Single-camera's methods of a specific camera, given its id number).

- Y-offset: given the id numbers of two cameras, returns the y-offset between the two cameras. — Point Inclusion (Isin): given a point data items, returns information on all cameras in the system that contains the item in their coverage regions, using Single-camera.''Isin for each camera. The information can be given either by a vector of cameras (or camera id numbers) containing the point, or a boolean vector indicating for each camera whether or not it contains the point.

- Polygonal Inclusion (Clip): given a quadrilateral (Dpolyline), clips it against the quadrilaterals representing the cameras coverage regions, using Single-camera::Clip for each camera.

2.1.4 Sip Camera Model (Sip-camera-model)

The Sip-camera-model class is used to handle information on a system of several cameras under the SIP. This class enables loading of camera parameters from configuration files.

2.1.5 Sip General Data (Sip-generaLdato)

Sip-general-data is a singleton aimed to provide access to general parameters and data, among which one can find access to the camera model related to the current scan, and the camera model related to the corresponding reference (methods Camera-Model() and Ref-Camera_Model(), respectively).

2.1.6 Windows (Sipwin)

A Sipwin object represents a rectangular region consisting of a variety of Sipdata items. As this object is the basic unit in the SIP, we endow windows with coordinate systems, and since a window is a geometric entity, it also endowed with WinGeomInclusion status.

2.2 Tasks 2.2.1 Reference Packing

Two kind of windows are processed during inspection: (a) top-down windows defined in aligned coordinates, and (b) on-line windows created around trigger in on-line camera coordinates. In either case, corresponding reference information need to be attached to on-line windows.

Top-Down (TD) Windows Management (Task-packer) Top-Down windows are given in aligned coordinates. For each execution graph, Task-packer creates camera coordinate TD window by first transforming the aligned window boundaries into camera coordinates. The correct transformation for the specific window is available to the Task-packer from its transformation Data-Source (ds-trans) computed for the line of window trigger. TD windows are given an id number. This id number indicates the origin of the window and will serve (at the final stage of merging information retrieved from several execution graphs) to identify windows with the same origin. TD reference information is transformed into on-line coordinates before being attached to windows and is held in that manner throughout the entire inspection processing.

Packing reference data "on the fly" (func-winref-adaptor) For on-line windows created around a trigger during inspect, relevant reference data need to be extracted from camera-based CEL reference. The on-line window boundaries determine the boundaries of the window of data to be extracted as follows: the on-line boundaries are first transformed into reference aligned coordinates. Then, for each camera in the reference camera model, those reference aligned window boundaries are transformed into camera coordinates, and the window is clipped against the camera coverage region. The camera for which the window fully falls within its coverage region, is chosen, and CEL data is extracted using the corresponding camera-based windowing query. Because CELs cannot be transformed from one coordinate system to another without a significant impairment to their structure, the extracted data is used as it is by a composite of functions comparing the on-line with the reference CEL data.

2.2.2 Single Slice Management (Task.test-manager)

Task-test-manager is traditionally responsible for the management and control of an entire execution graph (corresponding, of-course, to a single slice). In its current version, this task has an input and output queues of windows. The output queue consists of unresolved windows - which are windows that cannot be completely resolved within a single slice management and need to be directed to multi-slice processing. Task-test-manager is responsible to execute the functions attached to a window on that window, and to control their result.

2.2.3 Multi Slice Management (Task-multi-slice-manager)

Task-multLsUce-manager is responsible for the management and control of

windows arriving from several slices. The main difference between Task-test-manager and Task-multi-slice-manager is that the later is responsible for collecting windows having the same id (that is, windows with the same origin) as sub-windows of an overall window, and only then to Execute the corresponding multi-slice functions attached to that window on that window, and to control their result.

2.2.4 Split of Transformation Information (Task-split-trans)

Task-split-trans splits a transformation Cameras-Online —> Aligned-Reference given for one camera into the corresponding transformations for each of the cameras available in the camera model. Online - Reference transformation should take into account two transformations: first, the alignment transformation, transforming data from camera into aligned coordinates, and the transformation resulted from the registration process, which transform online data into reference coordinates. Note: The current registration procedure finds a match between online data coming from the central camera with reference data coming from the same camera. Thus, the transformation given to the Task-split-trans as input is not Cameras-Online -> Aligned-Reference as we would expect it, but Camera(l) -Online —>• Camera(l) -Reference, and all multiplications are done accordingly.

APPENDIX E

Abstract

This appendix briefly explains the function of the registration and the Line-Width measurements in the SIP. Line-Width is typically based on the CELs and the morphological skeleton, while the registration is typically based on the skeleton features only.

The registration

Most methods in the SIP compare a 'learned' panel to an actually scanned one (the 'inspect' phase). The function of the registration in the SIP, is to find linear transformation between points of the scanned panel to the reference coordinate system. The registration typically uses the output of the hardware morphological features extractor. It finds these transformations by using both features of the 'learn' and the 'inspect' different phases. Human intervention is not typically needed for this process, both for learning and inspecting.

The features are taken from the beginning of the panel, and are of limited number due to limitations in processing time determined by the computer speed, and the requirements from the SIP system. The numbers are typically around several hundreds.

It is also deals with limited movement, e.g. rotation. The shift form rigid transformation is preferably small (in the order of magnification up to 1% in x and y, and sheering angle preferably smaller than about 0.01 radian).

The registration is divided for two phases in the inspect. The first one is to find the initial transformation, and the next one, is to keep the panel in the best transformation during the scan. The second part is called the 'dynamic registration'.

The initial registration transformation method

learn process:

This process works on features of the learned panel, for illustration called group R.

Collect the first predetermined number of features from the beginning of the learned panel.

Sort the features of R according to x and y.

inspect process:

This process computes the registration transformation. It uses the features of R, and features which are collected at the inspection of the panel. These features are called herein for illustration, group I.

Collect the features in exact manner as in the learn phase.

Sort the features of I according to x and y.

Create a table with number of entries identical to the number of features in R, with entry N of the table representing feature N in R.

For each of the ordered features in I:

1. Push-back a copy of that features to all table entries which represent the features in R, which are in distance less the maximum miss registration (The table contains copies of features in I only).

For-each two pairs of features match between I to R combinations:

1. If the transformation is not rigid, move to next combination

2. If pairs are in S (set of pairs which will be defined below), move to next combination.

3. Find the transformation from I to R for this pair. if transformation is not in mis-registration maximum allowed angle or movement, move to the next combination.

4. Find all features down in the table that conform with the table entry representative feature in R using the transformation (if distance is small up to noise between transformed I feature and its R).

5. The features address which conform with the transformation are in a group - all pairs of combination of them are put to the set S

Move to next two pairs combination.

The registration transformation is the transformation which was computed with the maximum number of matches.

The Dynamic registration.

Dynamic registration is a method for tracing the most correct transformation during panel scan. Initial transformation is already known. A 'Matcher' process sends pairs of matched features to it. The core of its activity is computation of the best transformation which exists between two sets of matched points. The best

transformation is achieved using the least square method. Find the linear transformation which minimize the square distance of the pairs. Mathematically this can be described as shown in Equation 1.

Solving this minimization problem yields a simple solution, which is easy to handle. For each new matched pair some registers are update. Calculation of the best transformation is by using them.

Line-Width measurements

Line-Width process measures the width of the conductors, and the space between them. It finds violations in their measures by comparing the sizes of the 'learn' to the 'inspect' panel. The amount of the deviation is given by the user, using a graphical human interface that allows one to tell for each region in the panel, what are the used limitations. Line-Width is using the CELs and is done by software. It is officious due to its simplicity.

The measurements:

The measurement are taken in 4 directions: East, West, South-East, and South-West, i.e., they are typically taken each 360/8 = 45 degrees. The measure is taken perpendicularly to the morphological skeleton. A stream of CELs and Skeletons are the input for this method. The Skeleton typically holds the information of what is the perpendicular direction of width measurement. Each CEL or Skeleton feature belong to a certain state machine, in each of the directions. A state machine allows measurement if CEL, Skeleton, CEL combination found, otherwise no measure is output. The measurements uses the sub pixel CEL accuracy and tables exist for output information of the diagonals intersection with the cells.

Learn process:

The 'learn' coordinate system is divided to a small square boxes. The task of this phase is eventually to tell for each of this box, what is the minimum and maximum allowed measure. The knowledge is gathered by measuring the learned panel and putting the measures to the boxes. A statistical analysis is done to determine if a box is legitimate, and in a legitimate case to determine whether it is good or bad. In other cases usually minimal measure is put for a reference box. A reference box contains statistics also of measures near it in order to resolve the 1 to 2 pixels accuracy of features after

transformation.

Inspect process:

The inspect process is more simple - each measure is checked against the appropriate learn box, for being in the [min, max] range. The learn box is found by transforming the inspection coordinate to learn coordinate system, using the registration transformation.

EQUATION 1

$$\min\left(\sum_{i=0}^{N} \| \vec{p}_i - Tx\,\vec{Q}_i \|^2 \right)$$

where p_I matches Q_I belong to the sets P Q respectively.

APPENDIX F

Introduction

Appendix F will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 64 is a simplified block diagram illustrating inputs and outputs of a task_trigger_handler, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 65 is a simplified flow chart illustrating the function of a task_trigger_handler, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 66 is a simplified illustration of the run length structure of a task_trigger_handler where the map region is the bounding rectangle enlarged by two pixels on either side, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 67A is a simplified illustration of the snap rectangular regions cut according to the clusters defined in Fig. 66; and

Fig. 67B is a simplified illustration of the snap rectangular windows possessing the cluster, and a rectangle with RGB data, after processing of the clusters defined in Fig. 66.

This appendix describes a task existing in the SIP, named Task_trigger_handler. The purpose of this task is a primary fast sorting of information that originally exists in separated structures in the snap channel, and combining the relevant data into a single structure which is ready for futher processing. The task consumes two data sources - data source of Color_defects and data source of Snaps. The produced data source is of the type queue of windows. The operation of the task consists of three main parts : (i) on-line clustering of the Color-defect reports, (ii) cutting the appropriate rectangular areas which contain RGB-data out of the snap data source, and (iii) packing all of the relevant information into a form of a queue of sip windows. All of these sub-parts are applied on the data accumulated so far.

Reference is now made to Figure 64 which is a block diagram of a Task_trigger_handler and its inputs of a data source of Color defects and a data source of Snaps, and producing a queue of windows as an output.

Scope

Task_trigger_handler is applied on the input data sources while they are built. The resulting output, i.e., queue of windows, is treated by the Window_test_manager, and each window is directed for further processing, according to the associated test function. Therefore, Task_trigger_handler is the first station in the post process procedure, and its main goal is an efficient gathering of associated pieces of data needed for the post process.

Basic concepts

• Color_defect report : Color_defect report is a general name for reports that enter the SIP through the snap channel. These defects may be reported either by the Color Multi-Resolution Area Defect detector (COMRADD) which is described in Applicant's copending Israel Patent Application IL 131092 entitled "Optical Inspection System", filed 25 July 1999, or the Small Defect Detector (SDD), and may be associated with various types of surface defects, such as oxidations, stains, scratches, etc.

• Snap : Snap is a rectangular or modular-rectangular region that contains RGB-information. Color_defect report causes a recording of the RGB data inside a square area around it. The minimal size of the RGB- square is 48 X 48 (pixels).

• Data source : data source is one of the SIP building blocks, that carries data which is used by other SIP units. The reports that enter the SIP through the various channels are transformed into the format of data sources.

• Queue of windows :queue of pointers to Sipwin objects. Each Sipwin objects is a rectangle of a given size and location which includes a vector of Sipdata objects and a function to be executed by the window test manager.

•

Input/Output

Task _trigger_handler treats data sources that are originated in the snap channel during the scan. The input consists of two data sources: (i) data source of Color_defect reports and (ii) data source of Snap reports. The task produces a single data source - queue of windows. The specification of the consumed and produced data sources will be given in the following.

Consumed data sources

Task_trigger_handler consumes data sources of two types:

• Ds_array<Color_defect> - the source of reports to be clustered. Ds_array<Color_defect> contains an ordered (in (x,y)) vector of Color_defect reports. The Ds_array_lines keep the reports in a non-sparse way. A pointer to the beginning of each Ds_array_line is available. Each report contains its x-location (12 bits) and other specifications like type, slice of origin, and various characterizations (24 bits).

• Ds_array<Snap> - the source of the RGB data to be cut. Ds_array<Snap> contains an ordered vector of snap reports. Each snap report contains its x-location (8 bits), as produced by the hardware), and the red, green and blue values (8 bits each). In order to use the x-coordinate, it is typically converted from the 8-bit format into the regular 12-bit format. A pointer to the beginning of each ds_array_line is available.

Produced data source

Task_trigger_handler produces a single data source - a queue of windows.

Win_queue: the Win_queue is a data source of type Sipwin.

Sipwin: A general Sipwin object is a rectangular window that contains a vector of sipdata objects and a test function. The Sipwins that are produced by Task_trigger_handler contain two sip_data objects: sipdata_cluster and sipdata_RGB, and contains an appropriate test function - (presently it is an identity function).

sipdata_cluster: contains the reports that belong to the cluster.

sipdata_RGB: contains a rectangular piece of snap as an RGB image in a portable pixel map (ppm) format, and the coordinate of the top left point, relative to the global (0,0).

Method Structure

The method's spec of task trigger handler is the following:

The following steps are preferably performed:

1) clustering of Color_defect reports.

2) cutting the appropriate RGB rectangles out of the snap data.

3) packing all of the relevant information in a form of queue of sip windows.

The method is preferably performed on line.

The method may be applied only upon consumed data sources that had already been produced.

The method preferably is efficient, since it serves only as a tool for a primary fast sorting of information that prepares the data for further and deeper processing.

Reference is now made to Figure 65 which illustrates schematically the main blocks of the method, for a single loop over a bunch of scanned lines:

Clustering method

The clustering method finds clusters that consist only of points that are connected to each other. In principle the clustering is according to a given distance criterion. The input of the clustering part is an ordered set of points (color defect reports) and the distance criterion, and the output is a list of clusters and their bounding rectangles.

The clustering of points is done line by line, and uses the assumption that the color defects are ordered according to their coordinates.

For each point in the line it is determined to which (and how many) cluster/s it belongs. In the case of connectivity criterion in two dimensions, the maximal number of clusters to which a point can belong, N_max, is 2. If the distance criterion is general, N_max 4.

- N_max = 0 => a new cluster is opened with the current point.
- N_max = 1 => the point is added to the appropriate existing cluster.
- N_max = 2 => the point is added to one of the two clusters, and a merging procedure between the two clusters is performed.

The determination of N_max uses a "history buffer" which keeps the last clustered line, such that at every site which was occupied by a color_defect there is a pointer to the appropriate cluster (to which the color_defect belongs). This is use for a random access search for neighboring color_defects. There is an associated structure that keeps only pointers to occupied sites, for a more efficient update.

Each time that a certain number of lines (determined in advance) are passed, the clusters are checked to be ready. Each cluster, includes the color_defect points and the bounding rectangle.

Cutting of snaps

This part of the task, is performed by the Snap_cutter object, is responsible for the extraction of the RGB data in a given rectangular region, out of the full RGB-data existing in the data source of snaps. The Snap_cutter gets a set of sorted rectangles as an input (sorted according to the right-bottom point). These rectangles determine the regions in which the RGB data is typically filled. The outcome of this part is a set of rectangular RGB images, kept in a ppm format.

The method for cutting snap regions according to a given set of sorted rectangles is the following is performed in several steps.

• As a first step, a conversion of the set of rectangles to a run-length structure is performed. The elements in each run-length line represent the boundaries of the rectangles (with indication of location and if it is left or right boundary and a pointer to the appropriate cluster).

• As soon as the run-length structure is built, the RGB data is filled accordingly. The filling is done separately for each line, according to the following method:

1) Using the run-length data, one can determine the number of rectangles to which a segment between two consecutive run-length elements belongs.

2) A search in the data source of snaps for the pixels that belong to the segment is performed. The search uses the natural division of the data source of snaps into blocks of 16 pixels (all pixels inside such a block have the same value of x in 8 bits representation. As a first step of the search the relevant block is located, and then the exact pixel is found inside the block.

3) Fill the RGB data in the pixels on the segment between two successive run-length elements by continuously running over the appropriate line in the data source of snaps.

4) Keep a pointer to the last "used" element in the snap data source, so that it will serve as the starting point of the next search.

This method for cutting of snap regions is valid provided that the snap data is ordered, and the set of the rectangles is ordered.

Packing method

The part in which the packing is performed is quite straight forward. The method makes a merge between the data of the clusters and the data of the cut snaps. The packing is typically performed only for clusters for which the associated snap regions already exist. For each rectangular snap the associated cluster is located, and a new sip window (of the size of the snap) is opened. The cluster and the snap are written in the format of sipdata, as sipdata_cluster and sipdata_RGB, respectively, and added to the window. A test function is associated, and the window is added to the queue of windows and it is ready for further processing.

Summary

The task Task_trigger_handler gets as an input two data sources (data source of color defects and data source of snaps), and produces a queue of windows, that contains two sipdata objects - sipdata_cluster, and sipdata_RGB. Sipdata_cluster contains a list of color_defects that belong to a cluster, and some information about the cluster, like its type. Sipdata_RGB contains the RGB pixels inside a rectangular region that contains the color_defects of a particular cluster, ordered by the location, stored as an image in ppm format. The location of the top-left corner of rectangular image is given in an absolute coordinate system.

Several issues may be added or changed in the future, according to specific needs:

- Division into 3 tasks : Clusterer, Snap cutter, Packer.
- Clustering according to a general distance criterion.
- The shape of the cluster may be stored.

An example of the clustering method, provided below, may be understood with reference to Figures 66-67B.

APPENDIX G

Appendix G will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 68 is a simplified illustration of the general configuration of a snap area treated by the task_snap2ppm, constructed and operative in accordance with a preferred embodiment of the present invention; and

Fig. 69 is a simplified illustration of examples of configurations of snap areas not treated by the task_snap2ppm, constructed and operative in accordance with a preferred embodiment of the present invention.

Purpose

Task_snap2ppm is a tool that enables to separate snap data that contains several disconnected areas in which snap data exists into separate ppm files, each contains only one connected snap.

Input

The input of task_snap2ppm is the data source that contains the snap data.

Output

The output of task_snap2ppm are ppm-files, the number of which equals the number of separated snaps. Another file contains information about the location of each separate snap relative to the global (0.0), and it is written in a TCL format. The general configuration of the snap area which is treated by task_snap2ppm is illustrated in Figure 68.

Method

The method to separate the disconnected snap areas is the following:

Use a single loop to go over the data source of snaps, during which the separate output files are preferably either opened, or closed, or simply updated, according to the following rules:

If the current line is non empty while the previous line was empty a new ppm file is opened.

If both the current and the previous lines are non empty the buffer that contains the RGB data to be put in the current ppm file is updated.

If the current line is empty while the previous line was non empty the current ppm file is closed.

This simple method is valid under the following assumptions:

The connected snap areas are rectangular. The configuration shown in Fig. 68 is not treated by task_snap2ppm.

Each line contains RGB data that belongs to a single snap. The configuration shown in Fig. 69 is not treated by task_snap2ppm.

APPENDIX H

Appendix H will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 70 is a simplified illustration of a chain of blocks structure in a windowing reference management tool, constructed and operative in accordance with a preferred embodiment of the present invention; and

Fig. 71 is a simplified illustration of the indexing main structure of a windowing reference management tool, constructed and operative in accordance with a preferred embodiment of the present invention.

1. Introduction

I.I Scope

This tool for Windowing Reference Management aims to support efficient storage and retrieval operations of window data. The goal of this tool is to allow the storage of simple windowing data items which are inserted in order into an internal data structure, in such a way that allows their efficient retrieval.

Retrieving data efficiently is a common goal of many systems, especially those that call for a massive data processing and manipulation.

The motivation for the development of this specific tool arises from various application requirements arising from the inspection of BGAs and lead frames.

1.2 Glossary And Acronyms

• Template - an automated way to generate customized classes and functions.

• STL - Standard Template Library - a template-based library of generic C++ data structures and methods.

• Container - a class supporting data storage. Examples of STL containers include:

• vectors

• lists « queues and priority queues>>

• A container adapter class - classes that encapsulate an existing container, while providing a new user interface.

• A. function object - (template) object class with an operator () defined.

2. Concepts

A possible approach to the problem of efficient retrieval of windowing data from reference, in accordance with a preferred embodiment of the present invention is derived naturally from the observation that scanned image data is usually ordered; first,

by lines (y coordinate value) , then by the order within each line (the x coordinate value). Assuming data items have a key value according to which the items can be sorted, or ordered, the task of extracting information from windowing reference can be reduced to the task of extracting information from an ordered reference, as described herein. The basic prerequisites typically include:

• Data items have key values according to which items can be sorted (ordered). The key can be the value of the data item itself, in case of numeric data types. Keys are of a scalar data type, e.g., int, double, char, accessible by a function object. This requirement aims to provide a uniform access to the key, independently of data type definition. Thus, the order by which data items are ordered is subject to the key function supplied by the user.

• Data is given in order, that is, items are inserted in order into the local container either one by one, or an entire ordered container is inserted.

Thus the development of a reference tool arising from a preferred application needs to provide an efficient retrieval of data from containers that do not have an efficient method for data access: Various containers may not allow for an efficient data retrieval. One example of such container is the Chain (if Blocks container, a vector like container, for which memory is allocated in blocks, each containing a fixed number of elements). Blocks are allocated in chunks, each of which contains a fixed number of blocks (the Chain of Blocks structure is very similar to that of a linked list, see Figure 70). A direct (random) access to items in the chain is not possible. An adapter that can wrap the Chain of Blocks, while allowing it to retain its insertion functionality (e.g., push_back), with the benefit of an efficient retrieval of items is clearly used.

Next a design may be considered for a tool for indexing of ordered data; The main goal of such a tool is to support efficient retrieval of ordered data from reference. The reference can be external, e.g. a file, or internal — reference data located in memory. Moreover, the use of such a tool may be directly, or indirectly —via a container adapter, wrapping the container while allowing efficient access to its data items. To allow for efficient access of data, the indexing system is based on a number of indices, defined on construction. Each index corresponds to another sub range of possible key values, or bin. The bounds of the range of key values is defined by the minimum and maximum of possible key values over the items in the container to be indexed. The range is then sub

divided into equal size bins, whose number is specified by the used number of indices. Each index directs towards the data item whose key is the first to reside in the corresponding sub range.

2.1 Interfaces

The indexing tool is an STL/C++ independed library, therefore it can only be used as part of a C++ program.

2.2 Data

The following is a description of the integral data members involved in the indexing tool:

• unsigned int n_bins; The number of indices (bins) used for the indexing system.

• Bins Index', The indexing system: A vector of n_bins indices, or bins, covering the used range of possible key values, as defined by the following parameters:

• pair<Key::resulf_type, Key::result_iype> Xrange', The range over which the indexing system is defined, where Key is an object function allowing to access the data-item key. This range is defined by the minimum and maximum of possible key values over the items in the container to be indexed.

• Bins size. For each bin, the number of data-item keys that reside in that bin is stored. Note that only the index of the first data item whose key resides in that sub range need to be stored at each bin because data is ordered, thus the Bins size does not really reflect storage usage, but a potential one, i.e., it indicates the distribution of data items over the entire indexing system.

2.3 Error Handling

Errors are handled by return codes. Possible errors and their return codes are given in Table 1.

Table 1: Errors and Warnings Codes and Meanings

Function	Code	Error/Warning
Create index	Err(l)	Container data items are not ordered
load store	Err(2)	xternal device is not readable External device is not writeable

Create index	Err(3)	Empty Container - Nothing to index

3. Design Description

The indexing of ordered data tool is a template class; Two versions of this class exist: (a) A Memory Indexing object class whose indices direct into a Container in memory, and (b) A File Indexing object class whose indices direct into a file containing the data items to be indexed. Note that in the latter case, the indices are actually offsets in file. The File Indexing version is templated with a Key function object, whereas the Memory Indexing class is also templated with the Container. The following are the methods provided by both classes:

Construction

• Purpose - Construction and initialization.

• Inputs - A resolution parameter which defines the number of indices (or bins) to be used by the internal indexing system.

• Processing - The internal indexing system is initialized with the specified number of indices (bins).

• Outputs - None.

Create

• Purpose - Create the internal system

• Inputs - None.

• Processing - Define the range to cover using the minimum and maximum over key values over the items residing in the local container, calculate the range covered by each bin, and assign the value of the indices accordingly. For each bin, the corresponding index stores a reference to the first data item whose key is inside this specific bin range. The order of the data items is checked to verify that they are given in order. Note: the key values can only be accessed via the Key function object, with which the Indexing class is templated.

• Outputs - None.

Update

• Purpose - Update the internal system.

• Inputs - The new number of bins to use for resolution over that same range.

• Processing - Update the number of bins (indices) used by the internal indexing system, and re-create the indices accordingly.

• Outputs - None.

Query

• Purpose - Efficient retrieval of information out of the local container.

• Inputs - The query range.

• Processing - Using the indices, calculate the first and last bins in which data items satisfying the query range may reside.

• Outputs - A pair of bin numbers, between which data items are expected to satisfy the query criterion.

Storage Operations Serialization

• Purpose - Store the indexing system information.

• Inputs - The external device into which to direct the serialization and storage operation.

• Processing - Since the indices of the File indexing object are offsets, storing them is straightforward. For the Indexing in Memory case, serialization is a prerequisite. Consequently, the Memindexing has also the serialize and deserialize methods, converting its indices into offsets that can be stored into and loaded from and external device.

• Outputs - None.

Deserialization

• Purpose - Deserialize and load indexing information from external device.

• Inputs - The external device specification.

• Processing - Deserialize data, and load the indexing system information from the specified external device and restore it into memory.

• Outputs - None.

APPENDIX I

Appendix I will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 72 is a simplified illustration of fast retrieval of data via a windowing query, constructed and operative in accordance with a preferred embodiment of the present invention.

1. Introduction

I.I Scope

Windowing queries play an important role in image processing applications in accordance with preferred embodiments of the present invention. A window is a rectangular region of the image. A windowing query, or a region query, is the task of determining the part of the image contained in a given window, and providing its data. The window concept is a fundamental building block of many tasks in methods according to preferred embodiments of the present invention; windows that define regions of interest for detection, or the defect windows, are created around defects reported by the hardware.

The need to support efficient windowing queries is motivated by the following scenarios:

• Windowing queries during the Inspect scenario:

One method of inspection is based on the comparison of scanned data to reference data (data learned during a learn scan). In many cases however, only a part of the data (around regions of interest) is readily stored for future reference (via top-down windows). During an inspection scan scenario, on defect detection, there is a need to compare the window around the defect with the corresponding region as obtained at learn, even if that window is not a part of a previously stored window. Due to performance constraints, this comparison is preferably done as fast as possible.

• Windowing queries for visualization applications: A similar scenario occurs in applications of data visualization. At the heart of such applications stands the possibility to draw data on the screen at user's request. That is, by marking an area of interest, or a window, the user triggers the application to re-draw the data of that window providing higher level of details. Such interactive applications preferably access their data efficiently and accurately.

1.2 Applicability

The Windowing Reference Manager (WinRef) aims to provide an independent library tool, and to serve as part of the SIP. The WinRef is expected to be applicable to a variety of SIP tasks and data types and structures, and to allow for an efficient data access and retrieval of reference data, e.g., scanned data contained in hardware reports (such as CELs, Color CELs, Features, Snaps, etc.), or in data structures that does not provide an efficient (direct) access to their data items (such as the Chain of Blocks data structure).

2. General Description

2.1 Product Perspective

The Windowing Reference Manager is conceived to be a general purpose, independent library tool, that supports tasks and data types and structures contained in the SIP. Consequently, the target application may impose several restrictions that reduce the tool generality. Preferably the design is as generic as possible, so that its adaptability to other systems will be as simple as possible.

2.2 Product Functions

The fundamental functions of the Windowing Reference Manager needs to support are the insertion of window data items into an internal window data structure, and the efficient retrieval of data items satisfying a given windowing query.

2.3 User Characteristics

The user of the Windowing Reference Manager should be well acquainted with the concepts of the working environment, that is, of C++ and STL. A general knowledge of the SIP system is used for a full understanding of possible applications, and the constraints imposed on the tool design.

3. Functional Requirements

• Purpose - The Windowing Reference Manager (WinRef) is an object class aims to support storage and efficient retrieval operations of window data items.

• Methods The following are the fundamental methods of the WinRef class:

• Construction and Initialization.

• Data Insertion.

• Query operation; Allows for a fast retrieval of data according to some windowing query criterion. The query criterion provides the bounds of the used

windowing range.

• Serialization and Deserialization; The capability to store and retrieve information for backup purposes on an external device, such as file, or for transient purposes (e.g. transfer of data between computer systems via TCP/IP). At serialization, sufficient information is stored as to provide a reliable recovery (restore) of the whole window data (deserialization).

4. Interface Requirements The interface to the WinRef class is provided by its methods.

5. General Requirements

5.1 Performance

The tool performance requirements are dictated by those of the target application.

5.2 Error Handling Errors will be handled by return codes.

6. Design Aspects

The following are the design aspects imposed by the expected applications:

In accordance with a preferred embodiment of the present invention, a window may contain two basic types of data: (a) Simple data items such as CELs (Contour Elements), or Features, or, (b) Compound data types such as Polygons (e.g., connected components, or vectors). Simple data items provide local information over a pixel, or sub-pixel, thus, their amount in a single scan is large. Compound data types are fewer in number; nevertheless, each holds more detailed information, and their inter-relations are more complicated. The advantage of using simple data items over the compound ones is that the simple ones are always obtained in order from a scan: first, by lines (the y coordinate), and then by the order within each line (the x coordinate). This observation can be put to use by the future design. Thus, though being a major constraint forced by the target main application, the ordering constraint has the advantage of simplifying the implementation.

Providing that the WinRef tool is designated mainly to support efficient retrieval of simple data items, the order by which items are given can be used to store the items effectively for an efficient future extraction. In specific, if we further assume that data items have a key value according to which the items can be sorted, or ordered, the task of extracting information from windowing reference can be reduced to the task of extracting information from ordered reference.

APPENDIX J

Introduction

Scope

This appendix describes the processing of "balls" under the SIP in the ICP machine.

A "Ball" is a circular single component; collections of "balls" can be found in many applications.

In the ICP machine, "balls" processing is only done in pre-defined "balls" areas.

Such processing aims to test the components found in those areas against

design rules and some other pre-defined constraints, as will be discussed in details later.

Glossary

The following is a short list of terms and abbreviations used in the present appendix:

A "connected component" is a chain of CELs (Contour ELements), which can be

either open or closed. In principle, each vertex of such a chain holds information on the CEL

connecting this vertex to its neighbour along the chain.

A "generalized circle" is a structure defined over a single connected component.

A "generalized circle" holds information on the extent by which this circle can be considered a "good" one, based on both design rules and other pre-defined constraints.

The information stored in the generalized circle structure is such that a full and accurate report

on all defects detected on that component, can be derived from it.

An "isolated circle" is a circle which was found to be "good". See section "Circles Sampling" for the definition of a "good" circle suggested by the implementation described here.

"Balls" Configuration Requirements

SIP

The ICP/SIP implementation of balls learning procedure expects as input a list of descriptions

of radius ranges. A range is defined by *min* and *max*. values, within which a *nominal* radius value resides (the nominal radius value may or may not be provided as input).

In addition, some relevant test data is attached to each range. For example, to test circularity, a parameter relevant to the tolerance on fitness to circle is typically used.

Application

To go along with application needs, the specific format of data typically used by the SIP can be provided by the SIP Server, from various application formats. Note: the application typically only deals with diameter values, which are typically converted into radius values by the SIP Server.

The following combinations of parameter formats may be supported:

i. A list of *nominals*, cannot be null (derived from Application parameter: *balls_nominal_diameter*, conversion by multiplication with 0.5).

ii. A list of pairs *(min, max)* defining each range, can be null (derived from Application parameter: *balls_min_max_nominal_diameter*, conversion by multiplication with 0.5*)*.

iii. A list, or a single value of *percentage_radius_deviation* or *absolute_radius_deviation* (derived from Application parameters: *balls_percentage_diameter_deviation*, for which convertion is done by multiplication with 2.0, and *balls_absolute_diameter_deviation* - for which conversion is done by multiplication with 0.5).

iv. *adjust_to_nominal_radius* parameter (Expert parameter: *balls_adjust_to_nominal*), applicable in learning scenarios only, default is FALSE. If TRUE, radiuses of circles to be learned are adjusted to the closest matching nominal (if such exist).

v. A list, or a single value of test-data parameters:

Note: if both *abs* and *percentage* parameters are given, the stricter value of the two is

taken as the parameter.

- *maximum_percentage_nick_depth* / *maximum_absolute_nick_depth*, and *maximum_percentage_protrusion_depth* / *maximum_absolute_protrusion_depth*.
- *maximum_absolute_defect_area* / *maximum_percentage_defect_area*.
- *maximum_absolute_defect_length* / *maximum_percentage_defect_length*.
- *circle_fit_sensitivity* and *percentage_circle_fit*..

Note: All the above, but the last two parameters (which are Expert parameters: *balls_circle_fit_sensitivity*, and *balls_percentage_circle_fit*) - are Application parameters. No conversion diameter-to-radius is typically used for none of the above.

A list of *nominals* is preferably always provided. Providing a list of pairs (min, max) is optional.

If only *nominals* are given, at least one of the parameters defined above, is preferably provided. In that case, *min* and *max* values will be computed as follows:

min = *nominal* - *absolute_radius_deviation*, or,
min = *nominal* * (1 - *percentage_radius_deviation)*.

max = *nominal* + *absolute_radius_deviation*, or,
max = *nominal* * (1 + *percentage_radius_deviation)*.

In the above calculations, if both parameters (*absolute_radius_deviation* and *percentage_radius_deviation)* exist, the smaller value of the two is taken as the *min*, and the higher value of the two is taken as the *max*.

If a list of pairs (*min, max)* exist, the above calculation will serve to refine the boundaries of those ranges, only if the parameter *allow_range_adjust* is set to TRUE.

"Balls" Learning Scheme

Circles Sampling

Description of the "balls" learning procedure: The goal of this procedure is to process connected components found in a given "balls" area, and to produce all ball or circle information that can be derived from each component in that area. Note that this procedure processes information retrieved from one camera (slice) at a time.

- *Input*: A connected component.
- *Output*: A generalized circle.
- *Process*:

1. The (camera coordinates) points defining a connected component are served to compute a circle parameters: center and radius (possibly in aligned coordinates).

2. Each point is examined against a robust estimator of the circle parameters, and the "best" points are identified (this is done using an internal parameter *circle_fit_tolerance*, on which the user has no control). Those "best" points are used to refine the circle estimated parameters.

3. At this point, a circle undergoes a test for a match to nominal radius range. The circle's estimated radius is examined against the available radius range list; A match to range *i* is set if the circle's radius resides within the *i*'s range boundaries (*min*, *max*).

4. To complete the sampling procedure, some indicators of the quality of the estimation, and/or some other tests can be calculated and examined against internal tolerances. In accordance with a preferred embodiment of the present invention, a fitness to circle test is applied. The parameters *circle_fit_sensitivity* and *percentage_circle_fit* are taken from the radius range table, where the estimated radius is served as the access key. If the estimated radius does not match any radius range in the table, default values are used for the parameters.

A "good" circle is thus a connected component for which all test results are above given thresholds, and whose radius matches one and only one* radius range.

* A match to more than one radius is a miss-definition of the radius ranges. A warning on this will be given at loading, and on circle sampling (execution).

No defect will be reported, and the circle will be learned.

Analysis of Unified Circle Information

The above procedure is applied to any "balls" window. At the final processing stage, all "balls" windows are processed together to eliminate duplicate appearances of circles in cameras' overlap regions (matching of two circles in this case is based on matching their centers up to *max_alignment_shift* tolerance, taken from the corresponding Expert parameter, and on matching their radiuses up to some *radius_deviation* tolerance, which is an internal parameter).

The circles are classified into two groups: "good" and "bad". Each such group undergoes a different handling and testing.

"Good" Circles Processing:

First, duplicate appearances of same "good" circles are removed, then each "good" circle undergoes the following tests:

Area reduction test.

The Area reduction test verifies that the area of the connected component underlying that circle is approximately the same as the expected area of the circle itself. If there is a reduction in the circle's area which is above the *defect_area* parameter taken from the relevant entry in the radius_range table, a SHAPE_AREA_REDUCTION sip_defect is reported.

- Nick-protrusion test.

The Nick-protrusion test examines the points on the contour of the circular component. A preferred nick-protrusion test is described in Applicant's copending Israel application 131092.

Those points are supposed to reside at a distance equals to the radius of the circle from the circle's center. A Nick is reported if a point on the contour is located at distance smaller than (radius - inner_threshold) from the circle center. A protrusion is reported if a point of the contour is located at distance larger than (radius + outer_threshold) from the circle center. The parameters *inner_threshold and*

outer_threshold are taken from the entry in radius range table corresponding to the circle radius. They are derived from the Application parameters defining the criteria for nicks and protrusions, respectively.

Each Nick/Protrusion point is reported as NICK_PROT sip_defect.

- Clean area test.

This test verifies that the points on the contour of the circular component do not corrupt the "clean area" defined by a circle of radius *clean_radius* around the center of the sampled circle. At the moment, the parameter *clean_radius* is set to the value of *min_radius* of the radius range corresponding to the sampled circle radius. A point on the contour which violates the clean area criterion is reported as NICK_PROT_ON_BALL sip_defect.

If *adjust_to_nominal_radius* is TRUE, the radius of each learned circle is set to the matching nominal radius. As mentioned before, the adjustment is applicable only if nominal radiuses were defined.

The reference circles are stored in aligned coordinates.

"Bad" Components Processing:

Defected components which are instances of "good" circle, resulting from camera slicing are removed. Other defective components are tested according to their polarity.

- Same polarity as panel.

The length of the bad component is tested against *max_exposed_metal_length* threshold.

(The length of a connected component is defined as the largest diagonal of its bounding box.) If length exceeds the threshold, a SHAPE_LENGTH_EXCEEDS_THRESHOLD sip_defect is reported, at the point which is the center of that bounding box.

- Inverse Polarity: treat the bad circle as Pinhole.

To test the component as pinhole, we first find the circle within which this component resides. We use the radius of that circle to get thresholds from radius range table, and

test length of the pinhole against defect_length parameter, and its area against defect_area parameter. If one of those tests proves this pinhole to be above threshold, it is

reported as SHAPE_LENGTH_EXCEEDS_THRESHOLD, or SHAPE_AREA_EXCEEDS_THRESHOLD, and its center is reported as PINHOLE.

Inspection Scenario

In principle, the inspection scenario for circles is similar to the learning scenario. A delicate detail is the treatment of radius ranges; As in the learning scenario, a list of radius ranges is used, and the radius of an approximated online circle serves to access the list of radius ranges. Note that the radius range list provided can be the same list of radius ranges that was used during learning, or a new range list can be provided. If a totally new range list is used within which no radius of any reference circle resides, defects may encounter later on Excess-Missing tests. These test (see details below) are performed in "balls" inspection areas to verify that the appearance of circles on the panel matches the appearance of reference circles.

The inspection processing of connected components in "balls" window is the following:

- *Input*: A connected component.
- *Output*: Defect reports.
- *Process*:

1. Compute a (possibly aligned) circle (center and radius) out of the online connected component.
2. Identify the "best" points, and refine the circle estimated parameters.
3. Examine the circle's radius against the available radius range list.
4. Calculate the quality of the estimation (or other tests), and examine it against given tolerances.

5. After unifying circle information from all slices, classify into two groups and test "good" and "bad" circles as in the learning scenario.

6. Perform extra tests specific to inspection scenario:

The goal of the two following tests is to verify that any reference circle has an online circle matching it exactly (position and radius in our case), and vice versa. Matching in this context means that there is a reference circle with the same radius (up to the internal *radius_deviation* tolerance) and at the same center location (up to the Expert parameter *max_registration_shift* tolerance) as the online circle.

•

• Excess Test

This process prevents the appearance of unexpected circles, that is, connected components with circular shape, whose radius resides within the range of possible radii, but for which no matching reference circle exist.

• Missing Test

This process verifies that there is no reference circle which does not have any matching online circle. Any appearance of online circle matching a reference circle is marked. The number of appearances serves to identify those for which no match was found. Overmatching is also reported.

Extra "Balls" Services

"Balls" Statistics

Statistical information is useful for estimating the parameters typically used at learning (Design- Rule test). Therefore, statistical information is provided at the end of a learning scenario by a special task named *balls_measurenments*. The information is presented in two forms: an histogram representing the distribution of radius values *(a)* in the given radius ranges, and *(b)* in sub-ranges of fixed pre-defined size. Thus, the user can apply an initial learning process with only one pair of *min* and *max* values representing the range (0, Inf), then use the resulting histogram of radius distribution in fixed sub-ranges over that range, to determine an exact setting of the radius range list.

In addition, information on all circles that were learned (before being adjusted to nominals, if such option is used) are outputted; This includes the circles center and radius, and its fit - rounded to the closest fit level. The fit levels typically used are

determined by two parameters given to this task: *minimum_circle_fit* and *n_circle_fit_levels;* the task will accordingly output *n_circle_fit_levels* fit levels, from *minimum_circle_fit* and up.

Update "Balls" reference data

"Balls" reference data can be updated according to information retrieved from the application. The goal of this procedure is to allow the user to confirm the learning process. The user has the ability to select reference circles to be taken out from the reference, simply by providing

a list of points which are close enough to the centers of the circles to be removed (those points can be, e.g., provided by the application from 'mouse clicks' on that circle).

The update procedure produces a new reference data (in SIP format) out of the given reference data and the list of points of removal.

Defect information

Table 1:

DEFECT TYPE	DEFECT DESCRIPTION
EXCESS_CELS = 1	Deviation of points residing on the contour of a connected component (cels) from a circular contour.
SHAPE_MISMATCH = 10	Online shape exists but cannot be matched t any reference circle.
SHAPE_EXCESS = 12	Excess circle with the same polarity as the panel
SHAPE_MISSING = 13	Reference circle was expected at this point but was not found.
SHAPE_OVERMATCH = 14	More than one online circles match a single reference circle at this point.
PINHOLE = 15	Excess shape with polarity opposite to panel.
SHAPE_LENGTH_EXCEEDS_THRESHOLD	Excess shape with length (long diagonal axis) that exceeds threshold.

= 16	
SHAPE_AREA_EXCEEDS_THRESHOLD = 17	Excess shape with area that exceeds threshold.
SHAPE_AREA_REDUCTION = 18	Circle for which a massive reduction (over a threshold) in area has occurred.
OPEN_SHAPE = 19	A non-closed shape.
NO_RADIUS_MATCH = 50	A connected component for which no matching radius was found in the radius-range table.
NO_FIT_TO_CIRCLE = 51	A connected component which does not fit to a circle.
NICK_PROT = 40	A point of nick or protrusion (located on a circular contour).
NICK_PROT_ON_BALL = 41	A point of nick or protrusion inside clean area of a circle.
MISSING_DATA_FOR_COMPUTATION =201	No reference circles found in :balls" window, or Failure in creating connected components for that window, or No radius range test data was found.
UNKNOWN = 250	Unknown circle defect.

APPENDIX K

Appendix K will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 73 is a simplified illustration of a board whose image is to be analyzed; and

Fig. 74 is two simplified illustrations of a board whose image is being analyzed.

INTRODUCTION

A key problem in computer vision is the extraction of meaningful features from images. A popular approach is based on edge data.

A method for production of connected components composed of CELs (lists of edge elements generated using edge detection method producing single pixel contour elements or CELs) is now described.

OVERVIEW OF THE METHOD

General

The vector of CELCELs is arranged in a format called run length which means that the data is arranged in the sequence of scanning (sorted by y coordinate and then by x coordinate for every line). We can look at the run length format is a some kind of sparse raster format.

Reference is now made to Figure 73 which is useful in understanding a preferred embodiment of the present invention.

Raster to vector of edges image

Properties of CELCELs (basic cel)

A CEL (contour element) is a zero crossing vector which is defined on a pixel size rectangle that has four binary image pixels at its four corners.

The four pixels are assumed to be centered around the origin having coodinates P0,P1,P2 and P3.

The X and Y coordinates of the CEL are derived from the image coordinate system and are set according to the coordinates of the center of pixel P3.

In addition to the pixel grid coordinates there is also a sub pixel resolution coordinate system for each axis which is represented by an integer having n =

CEL_SUB_PIXEL_BITS bits. Thus the representable sub pixel coordinates are from 0 (representing 0.0) to 2n - 1 (which is SMALLER than 1.0 ,that is represented by 2n.

It is appreciated that 1.0 can not be represented in this number of bits since we typically need n + 1 bits of representation in order to be able to represent all integers from 0 up to 2n.

The CELReport structure preferably contain the following bit fields:

dir;

last;

first;

edge_code;

where:

edge_code:

defines the orientation of the CEL which goes from first_edge to last_edge:

Case 6 corresponds to a saddle situation(s) which is described below.

Case 7 is used for special cases of entities, which is discussed below.

first:

is the intersection point (in a cyclic coordinate system) of the cel with edge first_edge.

last:

is the intersection point (in a cyclic coordinate system) of the cel with edge last_edge.

The elements first and last are stored in a cyclic storage scheme.

The sub_pixel location of a point on an edge is determined by the edge on which that point lies. Thus a point on edge 0 with the coordinate 1 is above the point on edge 3 with a coordinate M - 1 The advantage of this scheme is that it allows us to represent ALL points in the sub-pixel with CEL_SUB_PIXEL_BITS bits since each vertex of the pixel coresponds to one edge with a coordinate of M (which CAN NOT be represented using only CEL_SUB_PIXEL_BITS bits) and to another edge with a coordinate of 0 (which is representable).

dir:

0 if the area of white material (elsewhere referred as negative DOG signis) to the left of the oriented cel line going from first_edge to last_edge.

1 if the area of white material (elsewhere referred as negative DOG signis) is to the right of the oriented cel line going from first_edge to last_edge.

Input and output

The input to the method is a vector of Contour Elements (or CELCELs) which are single pixel edge elements generated using edge detection method. The output of the method is a connected components object which is a collection of highly sampled contours of the shapes in the input image. The connected components are the first form of full vector representation of edges of the input image in the system.

Parameters

The method has two input parameters that control its operation:

* The first parameter is the Close connected segments or Edge connection flag that instructs the method whether or not to perform chains association along the edges of the window. Chain association is used for further processing in the system in order to perform operations on groups of chains that describe a specific entity in the image.

* The second parameter Enclose white regions or Connect dir flag ,which is relevant when the edge connection option is used, instructs the method about the way association is preferably performed.

Operation

In general the operation of the method can be summarized as reordering of scanline data to polygonal data. In addition to that the method m performs some other topological operations to add information to the understanding of the input edges data.

The input to the method is a vector of Contour Elements (or CELs) which are single pixel edge elements generated using edge detection method. The output of the method is a connected components object which is a collection of highly sampled contours of the shapes in the input image. The connected components are the first form of full vector representation of edges of the input image in the system.

APPENDIX L

Appendix L will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 75 is a polyline object illustrating the deviation parameter of a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 76 is a polyline object illustrating the deflection parameter of a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 77 is a graph illustrating the small_cel length parameter of a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 78 is a polyline object illustrating a 'quality assurance' mechanism in a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 79 is a polyline object illustrating an advanced 'quality assurance' mechanism in a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 80 is a graph illustrating the results of a performance test in a process for straight line approximation of connected components composed of CELs, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 81 is a graph derived from the graph of Fig. 80 by examining the situation where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style; and

Fig. 82 is a graph derived from the graph of Fig. 80 by examining the deflection angle where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style, and the deflection angle that produces minimum time vectorization.

INTRODUCTION

A key problem in computer vision is the extraction of meaningful features from images. A popular approach is based on edge data which, though useful itself, is better represented in a more manageable form. The type of description can be application dependent but is usually based on a combination of straight line approximations and if possible approximations with higher order curves such as circular arcs, conic sections, splines, and curvature primitives. Straight lines are standard inputs to model matching systems, but higher order features are becoming more popular since they permit more constraints on model identification.

In this Appendix, a method is described for straight line approximation of connected components composed of CELCELs (lists of edge elements generated using edge detection method producing single pixel contour elements [CELCELs] followed by edge linking). Among the properties of the method we find:

* *Efficiency* - Good representation of the data up to a desired tolerance along with substantial data reduction, so that higher order geometric representations can be easily fitted to the data. All this is done with relatively low time consumption.

* *Low dependence on parameters* - The method has a small number of parameters which affect its operation, some affect the time and memory consumption, and another independent one affects the output data quality (preservation of the shape).

OVERVIEW OF THE METHOD

General

At present the method is designed to work on highly sampled contours of a shape, constructed of single pixel contour elements (CELs) connected into chains. All the preparation processes are done by a series of binarization and raster to vector methods. In general we can look at the problem as getting a contour of a shape and producing another (compressed, more efficient) description as output.

The main goal of the method is to produce a good and efficient representation of the input data for later processing (reference for change detection ,shape recognition etc.). A "good representation" can be defined as a representation preserving the original shape up to a desired tolerance and removing redundant data as much as possible.

In general, the operation of the method is based on running on the input contour and searching for significant points or vertices for the description of the shape. The significance of a vertex is determined using angle difference criteria of an examined

CEL of the input contour with respect to its environment. When a significant vertex is found the method starts a strong examination of "preservation of the original shape" and improves the vectorization, by adding more vertices, to achieve a desired deviation tolerance. The method has also a simple noise reduction criterion that improves data reduction by ignoring small contour elements that add little information to the description of the shape. The following sections describe in detail the method's input, output, parameters and operation mechanisms.

Input and output

The input to the method is a connected components object which is a collection of highly sampled contours of the shapes in the input image. As mentioned earlier the input connected components object is a list of edge elements generated using edge detection method (producing single pixel contour elements or CELs) followed by edge linking method. The connected components are the first form of vector representation of edges of the input image in the system.

The output of the method is a polylines object which is a collection of contours of shapes of the input image. The polylines object is the basis for all shape modelling and shape comparison in the system.

Parameters

The method has three input parameters that control its operation:

* The most important parameter that controls the output data quality is the deviation (in pixels length units) tolerance of the output contour from the input contour. This parameter influences the smoothing of the input contour and also the data compression ratio.

Reference is now made to Figure 75 which aids in the understanding of one preferred embodiment of the present invention.

* The next parameter is the deflection (in degrees units) threshold, which is used to determine the importance of the examined vertex for the description of the shape. The main influence of this parameter is in the determination of the speed and memory consumption during the process. All the aspects of this parameter are discussed below in the performance test and results section.

Reference is now made to Figure 76 which aids in the understanding of one preferred embodiment of the present invention.

* The third parameter which is used to reduce noise is the small_cel length (in pixels length units) which is used to filter out short contour elements that have a highly unstable angular nature with respect to their neighbourhood. The small contribution of the small cels to the description of the shape allows us to automatically discard them ,as non important vertices ,from further analysis.

Reference is now made to Figure 77 which aids in the understanding of one preferred embodiment of the present invention.

Adaptors for the function parameters are used to improve efficiency.

1. The deflection angle is converted from degrees to radians
in order to save the transformation in the angle calculation
function of the examined segment.

2. The deviation and small_cel sizes are converted to square of
the size to save the sqrt() function all the time a distance
has to be calculated.

```
rdeflection = deflection * (PI / 180);
minus_rdeflection = 2 * PI - rdeflection;
sq_small_cel = small_cel * small_cel;
sq_deviation = deviation * deviation;
```

Operation

The main iteration of the method is performed on the connected components object where every chain of CELs is processed sequentially. The main loop first declares a new polyline in the output polylines data object and then executes the Vectorize_chain function, which prepares the desired result in the output data object. At the end of vectorization function the new polyline is closed and marked with some additional system data for further use.

Execute code

```
// run over all chains and vectorize each one of them
 for ( unsigned int i = 0; i < c_components->size(); ++i )
 {
  data->BeginPolyline();
  VectorizeChain( data, (c_components)[i],
rdeflection,
```

```
sq_small_cel,
sq_deviation);
 data->EndPolyline((c_components)[i].IsClosed(),
(c_components)[i].Id(),
(c_components)[i].MtrlType());
 }
```

Vectorize chain

The Vectorize_chain function receives as parameters, pointers to the input and output objects and the three operational parameters described above. The function simply runs on the input chain (of CELs) and looks for "significant" vertices for the description of the shape. If such an important vertex is found, it is added to the output object.

It is appreciated that some exceptional cases exist, e.g. an empty chain of cels that can not be processed so the function exits. Also if ,for some reason, the chain includes only one vertex, there is no need for processing so the function inserts the first vertex into the output object and exits.

Another convention that should be mentioned here concerns the geometric properties of vertices and segments of connected components and polylines in the system:

* Angle(i) is the angle of i'th segment (first endpoint of CEL or line) pol[i]-->pol[i+1]

* Length(i) is the length of i'th segment (first endpoint of CEL or line) pol[i]-->pol[i+1]

* Sq_length(i) is the Squared length of i'th segment.

After the first special cases checking the function initialises the angle variable to the angle of the first segment, and resets an index for further use. At this time we are ready to start running along the chain and find "important" vertices. For each vertex we have the segment angle and length and the angle difference to the entrance angle of the examined group of segments (a group of segments is defined between every two "important" vertices in the output polyline).

If the examined CEL's angle_difference is larger then the deflection threshold it is considered an important vertex for the description of the shape.

When examining this "vertex importance" special cases are considered for segments with north azimuth that jumps from small angles (around 0 degrees) to large angles (around 360 degrees). Additional part of the question includes the "small cels" filter which bypasses the cases where a segment (a very short CEL) is accompanied by very large deflection angles on both his sides that crosses the deflection threshold and adds vertices which do not add important information to the description of the shape and substantially reduces the compression ratio.

If it was found that the examined vertex is "important for the description of the shape" it is typically inserted it into the output data structure. As a result of this insertion a new line will be created between the previously inserted and the newly found vertices. For some reasons it could happen that the newly created segment substantially deviates from the original contour. This incident is common along low curvature and smooth contours. In order to avoid the loss of information caused by these cases, the proposed vectorization are typically rechecked for deviations with respect to a given tolerance. This checking is done by the Improve_segment function before adding the new "important" vertex into the output object.

In general, the Improve_segment function checks that the proposed segment does not deviate from the original contour more then the permitted distance, and adds additional vertices if used. After improvement is done the new proposed vertex is inserted into the output vector and some more data is updated to continue the iteration. After the whole chain is processed we typically conclude by inserting the last vertex, of course with improvement of segment before insertion. The detailed operation of the improvement function is described below.

```
Vectorize_chain code
{
unsigned improve_sample = 2;  // index
if ( ipol.size() == 0 ) return; // connected component contains no vertices

data.push_back( ipol[0] );
if ( ipol.size() == 1 ) return; // connected component contains only 1 vertex
double  entrance = ipol.Angle(0);
unsigned startind = 0;
```

```
for (unsigned i = 1; i < ipol.size()-1; ++i )
{
 // Get azimuth and square length of the tested segment
 // and calculate angle difference.
 double angle   = ipol.Angle( i );
 double diffangle = fabs(angle-entrance );
 double sq_length = ipol.Sq_length( i );
 // Check significance of current vertex
if ((diffangle > rdeflection) andand (diffangle < minus_rdeflection) andand
(sq_length > sq_small_cel))
 {
  // About to add a vertex
  // Improve the new segment sapling
Improve_segment(ipol, startind, i, improve_sample,
  sq_deviation, data, cel_touch);
  data.push_back( ipol[i] );
  entrance = angle;
  startind = i;
 }
}
// About to add a last vertex
  // Improve the new segment sampling
Improve_segment(ipol, startind, ipol.size()-1, improve_sample,
sq_deviation, data, cel_touch);
data.push_back( ipol[ipol.size()-1] );

return;
}
```

Improve segment

The decision rules used to define "important vertices" along the chain has some weakness in detecting very low deflections of the contour, a problem that could result in a low quality description of the shape. As a result of this, and other possible

deficiencies, some mechanism for quality assurance of the vectorized result is typically used. The mechanism used is a very simple examination of the difference between the proposed vectorized resulting segment and the input chain of cels which it is meant to replace.

The Improve Segment function works on a predefined segment of the input connected chain of cels (starting and ending indices of the desired cels in the chain, defined by the upper level Vectorize Chain function). The examination is performed along the whole input segment and the vertex with maximum distance is pointed out. If the distance is larger then the allowed parameter then the most distant vertex is also added to the output polyline.

The measurement of the distance between the examined cel the proposed output segment is based on the measurement of the area of the triangle between the two proposed end points and the point on the segment of cels which is bounded by these points, as described in Equation 2.

The desired length is equal to (2*Area of the triangle)/(length of the base) where the length of the base is the distance between points p0 and p1 (Fig. 78). Since the base length is constant during the whole improvement iteration we can first calculate the base length term and save some calculations. Since the improvement iteration adds another touch to every cel in the input chain there is a big motivation to reduce the number of measurements. In the improvement iteration on the chain of cels a sampling parameter is used which skips a desired number of CEL distance measurements. The assumption behind this sampling parameter is that there is a relatively smooth contour of CELs between the starting and ending points that for some reason was not detected by the vectorize chain function and any high curvature features are not lost so a sparser quality checking is allowed.

It was mentioned hereinabove that the Improve segment function is preferably called before insertion of every new vertex into the output polygon in order to keep the right order of vertices of the output shape. Since the improve segment function inserts vertices itself preferably are called recursively call it before inserting additional vertices and also call it after the insertion in order to refine the output polygon according to the desired quality parameter.

Improve_segment code

```
{
if ((endind-startind) < improve_sample) return;
Dpoint2 p0 = ipol[startind], p1 = ipol[endind];
double area2, sq_dist, maxdist = 0;
double sq_seg_length = p0.Sq_dist( p1 ); // base length
unsigned maxind = 0;
for (unsigned j = startind; j < endind; j+=improve_sample)
{
 area2  = (p1.y - ipol[j].y)*(p0.x - ipol[j].x) -
     (p0.y - ipol[j].y)*(p1.x - ipol[j].x);
 sq_dist = area2*area2/sq_seg_length;
if (sq_dist > maxdist)
 {
  maxdist = sq_dist;
  maxind = j;
 }
 cel_touch++;
}
if (maxdist > sq_deviation)
{
 Improve_segment(ipol, startind, maxind, improve_sample,
 sq_deviation, data, cel_touch);
 data.push_back( ipol[maxind] );
 Improve_segment(ipol, maxind,  endind, improve_sample,
 sq_deviation, data, cel_touch);
}
return;
}
```

3. PERFORMANCE TEST AND RESULTS

When trying to estimate the performance of the method we preferably define the relevant quality parameters which are:

* preservation of the shape

* compression ratio of redundant data

* time consumption in the specific scenario

Some quality parameters are very rigid and are dictated by the user application, for example the preservation of the shape, while other parameters influence the operation of the method but not the output result that the user receives.

The compression of redundant data is the ratio between the number of contour elements in the input connected components data object and the output number of vertices in the output polylines object while the time consumption was measured using a timer object.

It is appreciated that the two last quality parameters can influence the operation of the whole system on conditions where time is a very important parameter that should be satisfied on some operational specifications, For example an unwise selection of the deflection parameter in learn scan can result with a low compression ratio which can degrade the performance of test functions in inspect scan.

For the performance test of the method a sample panel is used containing about 93000 contour elements (CELs) that are connected to contour chains using a separate connected components method. For this panel, the influence of the various input parameters on the output quality is tested.

In every test phase it is assumed that the preservation of the shape parameter is rigidly dictated by the user (this quality parameter is directly controlled by the deviation input parameter described above) and the deflection parameter is varied while testing its influence on the compression ratio and the time consumption of the method execution. During the whole test the small_cels length parameter is typically fixed to 0.3 pixel and the improvement resampling index parameter is typically fixed to 2 (every second CEL is examined in the improvement iteration). The whole set of deflection variations is tested using three deviation conditions: "tight" (0.1 pixel deviation) "medium" (0.3 pixel deviation) and "loose" (0.6 pixel deviation). The whole test is carried out on a Intel Pentium pro (dual processor) computer which was completely devoted for the analysis with out any users or processes.

The test results are displayed on Figure 80. On the results we first notice the inverse relation between the compression ratio and the time consumption as changing

from tight to loose vectorization (tighter vectorization consumes more time and delivers lower compression ratio). We can also see the logarithmic nature of the compression ratio as a function of the deflection parameter and that every vectorization style has an a constant compression level after some deflection limit, which implies that the final compression ratio is mainly controlled by the deviation parameter.

Observation of the time consumption data reveals the existence of a minimum value of execution time. Examining the code it can be seen that the time consumption is directly related to the number of CELs in the Connected components structure because the Vectorize chain function touches every CEL once (constant time consumption) and the Improve segment function touches the CELs some more times according to the necessary improvements. The overhead of the Improve segment function is added in any case no matter if the improvement is found necessary or not during a specific iteration.

In the performance tests done with the function, an improvement skip parameter of 2 is typically used which means that the number of touches added by the Improve segment function is 0.5 of the number of CELs in the case where no improvement is needed. This parameter remains constant during the whole operation test and its influence was not tested.

After all these explanations it can be concluded that the minimal number of CEL touches (of any kind) done by the function is 1.5 times the number of CELs in the input Connected components object one touch by the Vectorize chain function and at least 0.5 touch (in the usually used configuration) by the Improve segment function.

The "real" number of CEL touches was measured by counting the overall number of touches performed by the Improve segment function and is found to vary significantly as a function of the deflection parameter. For example if using the "0.3 pixel deviation" vectorizer and setting the deflection parameter to 10 degrees the result shows that the number of touches performed by Improve segment is 0.74 (resulting with 1.74 touches) of the number of CELs in the input. The excess touches (above ratio of 0.5) are caused because of the recursive calls to the Improve segment function.

Changing the deflection parameter to 20 degrees resulted with an overall CEL touches of 2.84 (1 + 1.84) of the CELs number in the input. A deflection parameter of

30 degrees for example causes an overall ratio of 3.34 because of many recursive improvement operations.

The overhead value of 0.508 is found when used a value of 7 degrees is used a the deflection parameter which means that almost no recursive operations were used in this iteration.

When the deflection parameter value is reduced to 3 degrees the resulting overhead is found to be 0.32 which means that not all the CELs are inspected by the improve segment function. This result is found because the improvement is requested for segments with length smaller then 2 CELs (see the first line in the Improve_segment code).

It can thus be concluded that the overhead touches result, which is controlled by the deflection parameter, strongly affects the time consumption of the function.

It is appreciated that the whole performance analysis was performed using a fixed small cel filter of 0.3 pixel. The influence of this parameter on the operation of the method is considered minor on the basis of the accumulated operational experience with the system.

CONCLUSIONS

When coming to select parameters for the operation of the method, if it is assumed that the deviation parameter (or the requested preservation of the shape) is dictated by the application, it can be concluded that the deflection parameter is the main control on the compression ratio and the time consumption of the method.

In the currently common use of the method (preparation of reference vector data for shape change detection) the main goal is to supply the maximum compressed result without unnecessary usage of time. This is especially true in the preparation of reference data of areas which can not be characterised during learn scan, forcing the necessity to perform vectorization during inspection scan.

In the following graphs it can be concluded that the operational recommendation for parameters setting of the method. The first graph shows the maximum compression ratio as a function of the deviation tolerance parameter. This is the first piece of information that has to be considered when coming to design vectorization for a specific application.

From the graphs in figure 80 it is obvious that requesting a tighter vectorization (better preservation of the shape) will cost extra time and reduce compression ratio, so preferably figures out the optimal shape preservation for a given application to meet time consumption constraints.

Figure 81, which is derived from figure 80, is a graph created by examining the situation where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style (or for a specific deviation parameter setting). The nearly linear nature of this function in the examined range (between 0.1 to 0.6 pixel deviation) is evident from the graph. The performance of the method was not tested beyond this range because in the 0.1 pixel deviation tolerance condition was found to produce a highly redundant result for our application and the 0.6 pixel condition produced poor and useless "preservation of the shape" results.

In accordance with a preferred embodiment of the present invention, a deviation value of 0.3 pixel which is used (with typical defect tolerances of several applications). If for some applications the proposed vectorization is not adequate for the preservation of the original shapes, the parameter may be changed, remembering that the maximum compression ratio produced by the vectorizer will change accordingly.

The next piece of information that has to be considered is the deflection angle that has to be used for every deviation setting that will result in an optimal vectorization (best compression with no unnecessary time consumption).

Figure 82, which is a graph derived from figure 80 was created by examining the actual deflection angle where the compression ratio (as a function of deflection angle) becomes constant for every vectorization style (or for a specific deviation parameter setting). The nearly linear nature of this function is also evident from the graph.

In addition to the optimal compression recommendation I showed in the graph a deflection angle that will produce minimum time vectorization as derived from the local minima in the time consumption graphs for every deviation parameter setting.

Even though there is a "minimum time consumption" situation for every vectorization style (or deviation tolerance) it is typically not recommended to perform such "minimum time consumption" vectorizations ,even with very tight time constraints scenarios, because of the poor output compression ratios, that may have to be paid later in the operation of further methods (such as shape change detection).

It is appreciated that the whole performance analysis was performed using a fixed small cel filter of 0.3 pixel and an improve resampling step of 2. The influence of these parameters on the operation of the method is considered minor on the basis of accumulated operational experience with the system.

It is anticipated that automatic tools may be able to automatically produce the necessary data for calibration of the parameters for optimal vectorization as dictated for every application the system will meet. In general the tool has to produce the two dimensional functions of compression ratio and time consumption as a function of the deviation and the deflection parameters.

EQUATION 2

$$2 * area = \begin{vmatrix} p0.x & p0.y \\ p1.x & p1.y \\ pi.x & pi.y \end{vmatrix}$$

(11 operations: 6 multiply, 3 subtract, 2 add)

which can be reduced to:

2 * Area = (p1.y - pi.y) * (p0.x - pi.x) - (p0.y - pi.y) * (p1.x - pi.x)